1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 9.Sep.02	3. REPORT TYPE AND DATES COVERED DISSERTATION

4. TITLE AND SUBTITLE
SOLUTION APPROACHES FOR THE PARALLEL IDENTICAL MACHINE SCHEDULING PROBLEM SITH SEQUENCE DEPENDENT SETUPS"

5. FUNDING NUMBERS

6. AUTHOR(S)
MAJ ANDERSON BRADLEY E

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
INDIANA UNIVERSITY BLOOMINGTON

8. PERFORMING ORGANIZATION REPORT NUMBER
CI02-622

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
THE DEPARTMENT OF THE AIR FORCE
AFIT/CIA, BLDG 125
2950 P STREET
WPAFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Unlimited distribution
In Accordance With AFI 35-205/AFIT Sup 1

DISTRIBUTION STATEMENT A:
Approved for Public Release -
Distribution Unlimited

. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

20021029 020

14. SUBJECT TERMS

15. NUMBER OF PAGES
112

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

SOLUTION APPROACHES FOR THE PARALLEL IDENTICAL

MACHINE SCHEDULING PROBLEM

WITH SEQUENCE DEPENDENT SETUPS

Bradley E. Anderson

Submitted to the Faculty of the University Graduate School
in Partial Fulfillment of the Requirements
for the Degree
Doctor of Philosophy
in the Kelley School of Business
Indiana University

September, 2002

ABSTRACT

The setup scheduling problem is the problem of determining the sequence of multiple products produced on one or more resources/machines. The sequence-dependent setup scheduling problem is more difficult than the setup scheduling problem and extends it by incorporating different setup costs or times for each product, based on the product for which the resources were set up last. When producing multiple products on limited-capacity resources, minimizing the earliness and tardiness of product delivery is an important scheduling objective in the just-in-time (JIT) environment. Items produced too early incur holding costs, while items produced too late incur costs in the form of dissatisfied customers.

This research compares the efficacy of a new network-based mixed-integer programming (MIP) formulation to an existing mixed-integer formulation for both the tardiness and the earliness/tardiness problems. An effective ET heuristic is also developed for earliness/tardiness problems too large to be solved efficiently by the MIP formulation. The presented MIP formulation provides a unique and useful method of conceptualizing and modeling a practical, yet difficult, problem within industry.
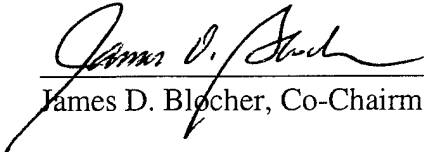
This research shows that the new MIP model is much more efficient in terms of computation time for multi-machine problems than another known generalized formulation of these problems. The structure of our model, which adapts a network-based traveling salesman problem (TSP) structure to multiple machines, enables it to function as a new benchmark for future model improvements. The problem structure in our model enables CPLEX MIP software to solve problems with a greater number of machines increasingly faster. The mixed-integer nature of the formulation allows
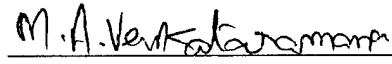
companies to solve this class of problems with any one of a number of commonly available integer programming software packages. The heuristic algorithm provides another effective tool for solving larger problems of this class where the MIP formulations become computationally too difficult to solve in a reasonable amount of time.

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government

ACCEPTANCE

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements of the Degree of Doctor of Philosophy in Business.

James D. Blocher, Co-Chairman

M.A. Venkataramanan, Co-Chairman

Doctoral Committee

S. Christian Albright

August 26, 2002

Kurt M. Bretthauer

ii

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1. INTRODUCTION AND MOTIVATION

Developing production schedules is a difficult job manufacturing firms face every day, as consistently generating efficient schedules can result in substantial improvements in productivity and time reductions. When producing multiple products on limited-capacity resources, minimizing the tardiness, or earliness and tardiness, of product delivery are important scheduling objectives.

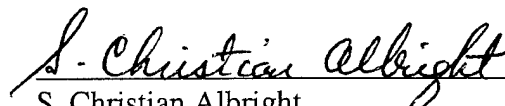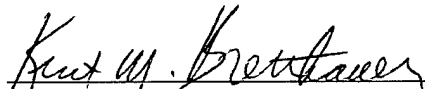The Setup Scheduling Problem (SSP) or Changeover Scheduling Problem (CSP) is the problem of determining the sequence of multiple products produced on one or more resources/machines. One common simplification made in studies of shop scheduling is that setup times (also known as changeover times) are independent of the sequence in which the jobs are processed. Setup times are then either assumed to be part of the operation time of the jobs or are ignored altogether. Although this assumption may be reasonable for some manufacturing systems, evidence suggests that in many systems sequence dependent setup times are the rule rather than the exception. Panwalker et al. (1973) found, in a survey of industrial schedulers, that 70% of the schedulers reported at least some operations they schedule are subject to sequence dependent setup times. Thirteen percent of the schedulers reported that all the operations they schedule are subject to sequence dependent setups. Panwalker et al. also reported that industrial schedulers overwhelmingly named meeting due dates as their most important scheduling criterion. The other choices were minimizing processing time, minimizing setup time, and minimizing in-process inventory cost.

1

The sequence-dependent setup scheduling problem (SDSSP) is more difficult than the SSP problem and extends it by incorporating different setup costs or times for each product, based on the product for which the resources were set up last. This problem naturally arises when a machine is producing parts with the same shape, but different colors. A small amount of residual white pigment has little impact on the shade of black; however, even a small amount of black pigment can notably change the shade of white. This causes off-color material waste or the equipment cleaning time between runs of one color and another to depend on the colors themselves. Less off-color material is generated or less machine cleaning time is required when the machine changes from white to black, as opposed to black to white. Despite the apparent practical importance of the problem, little work has appeared on scheduling to meet due dates in the presence of sequence dependent setup times. Rather, the work on scheduling in this environment has tended to focus on minimizing the total setup time (or cost) or minimizing the makespan of a set of jobs.

The problem addressed in this research is sequencing to minimize the total earliness and tardiness of a set of jobs in a multi-product, multi-machine version of the SDSSP (also called the N/M/ET problem with sequence dependent setups). The problem of sequencing to minimize total tardiness only (also called the N/M/T problem with sequence dependent setups) is also addressed as a special case of the N/M/ET problem. In particular, the efficacy of a new network mixed-integer programming (MIP) formulation to obtain optimal schedules is compared to an existing formulation. An ET heuristic is also developed to solve larger problems efficiently in this environment. The

existence of non-negligible sequence-dependent setup times and multiple machines is what makes this scheduling activity non-trivial.

This paper consists of 8 sections. The next section (2) contains the Problem Description. Section (3) is a Literature Review of pertinent research. The Mathematical Problem Formulations and Heuristic are explained in section (4). Section (5) discusses the Research and Experimental Design. Section (6) examines the Experimental Results, and Conclusions are outlined in section (7).

SECTION 2. PROBLEM DESCRIPTION

The earliness and tardiness (ET) scheduling problem, also known as the minimum absolute deviation problem, in its simplest form is scheduling N jobs to minimize the sum of differences between job completion times and due dates. The objective of the ET problem is well suited to JIT production control policy where an early or late delivery of an order results in an increase in the production costs.

For each job j to be done, there is a required processing time p_j, a due date d_j, and a setup time s_{ij} incurred when job j follows job i in the processing sequence. The sequence of jobs is defined as $Q = \{Q(0), Q(1), ..., Q(N)\}$, where $Q(j)$ is the index of the jth job in the sequence and $Q(0) = 0$. The completion time of the jth job in the sequence is

$$c_{Q(j)} = \sum_{k=1}^{j} \left[s_{Q(k-1)Q(k)} + p_{Q(k)} \right]$$

the tardiness of the jth job in the sequence is

$$t_{Q(j)} = \max\{0, c_{Q(j)} - d_{Q(j)}\}$$

and the earliness of the jth job in the sequence is

$$e_{Q(j)} = \max\{0, d_{Q(j)} - c_{Q(j)}\}.$$

We want to find a sequence that minimizes the total earliness and tardiness of the jobs. Total earliness and tardiness for a sequence Q is

$$T_Q = \sum_{j=1}^{N} (t_{Q(j)} + e_{Q(j)})$$

Because the use of both earliness and tardiness in the objective function creates a nonregular performance measure (Baker and Scudder, 1990), it is not sufficient to consider only permutation schedules. Schedules with inserted idle time must also be considered. A performance measure is regular if the scheduling objective is to minimize the objective function and this function can increase only if at least one of the completion times in the schedule increases. This is not the case for the ET problem, whereas the tardiness problem is a regular measure and requires no inserted idle time for optimal solutions. The tardiness problem is a special case of the ET problem where earliness is not included in the objective function and inserted idle time is always zero. Even the simple tardiness problem is NP-complete, and the number of potential solutions is very large for problems with even a modest number of jobs to be sequenced.

SECTION 3. LITERATURE REVIEW

This research addresses the ET scheduling problem with sequence-dependent setup times and the tardiness scheduling problem with sequence-dependent setup times. A brief review of literature on the work in this field and related papers is presented. The first section is a brief summary of ET sequence-dependent research and how this research fits into the research area. The second section discusses work on the tardiness problem with sequence-dependent setups. The third and final section discusses some related research that is not sequence-dependent.

3.1 Earliness/Tardiness Sequence-Dependent Literature

A review of the literature shows very few published articles address scheduling jobs to minimize total earliness and tardiness with sequence-dependent setup times. Almost all ET articles assume that setup time is either negligible or sequence-independent. Sequence-dependent setup times are very important to some industries, and Baker (1974) states a sequence-dependent setup time is a defining characteristic of classical job-shop scheduling.

Only five published articles were found in this area (see Figure 3.1). Two of the articles, Coleman (1992) and Chen (1997), address the single machine problem and are both optimal methods. Coleman (1992) first developed a mixed integer formulation for the ET problem with sequence-dependent setups. Chen (1997) utilized a dynamic programming algorithm, but he addressed only two types/batches of jobs on a single machine. The other three of the five articles in this research area address multiple different machines with sequence-dependent setups. Two of the articles, Balakrishnan et

al. (1999) and Zhu and Heady (2000), provide an optimal model. An article by Sivrikaya-Serifoglu and Ulusoy (1999) employs a heuristic method. Balakrishnan et al. (1999) developed a formulation for the multiple machine ET problem based on the special case that setup times for sequential jobs on any machine satisfy the triangular law of inequality. Their formulation also requires that i be less than j for any two sequentially scheduled jobs, where i and j are job identifiers. Zhu and Heady (2000) extended the single machine Coleman model to multiple machines. Sivrikaya-Serifoglu and Ulusoy (1999) developed two genetic algorithms, one with a tailored crossover operator and the other with none. None of the algorithms of the five noted articles were compared with solution methods or models by other authors.

	Single Machine	Multi-Machine Identical	Multi-Machine Different
Optimal Model	Coleman (1992) Chen (1997)	**THIS RESEARCH**	Balakrishnan et al. (1999) Zhu and Heady (2000)
Heuristic Model		**THIS RESEARCH**	Sivrikaya – Serifoglu & Ulusoy (1999)

Figure 3.1 – Sequence-Dependent Setup Scheduling Literature for the ET problem

The Zhu and Heady formulation can directly be used to solve the special case of multiple identical parallel machines, and is used as a point of comparison for this research. Zhu and Heady (2000) present average computational results for problems with

5, 6, 7, 8, and 9 jobs and 1, 2, and 3 machines. They ran 6 replications for each combination of job and machine quantity on a 66 MHz personal computer with a 486 processor. The times varied from an average of 1.64 seconds for the 5 job, 1 machine problems to 5397.33 seconds for 9 jobs and 3 machines.

3.2 Tardiness Sequence-Dependent Literature

A number of tardiness problem articles have been published that include sequence-dependent setups, but a very small number address an optimal method. Only Ragatz (1989) and Tan et al. (1999) were found to include an optimal solution method, where branch-and-bound was used to solve the single machine problem (see Figure 3.2). No literature was found that addresses utilizing an optimal solution method for multiple machines.

Ragatz (1989) developed a branch-and-bound procedure for the minimum tardiness problem with sequence dependent setup times. The bounds used in the procedure were weak, and the performance of the procedure was highly dependent on problem characteristics. Tan et al. (1999) compared the heuristic methods of genetic search, simulated annealing, and random-start pairwise interchange with Ragatz's optimal branch-and-bound method. Their research suggests simulated annealing and random-start pairwise interchange are viable for large combinatorial problems, while branch-and-bound may be preferable for smaller problems where an optimal solution can be reached.

	Single Machine	Multi-Machine
Optimal Model	Ragatz (1989) Tan et al. (1999)* (* heuristics also)	**THIS RESEARCH**
Heuristic Model	Rubin & Ragatz (1995) Lee et al. (1997) Sun & Noble (1999) Sun et al. (1999) Franca et al. (2000) Pan et al. (2001)	Lee & Kim (1993) Kim et al (1996) Parthasarathy et al. (1996) Lee & Pinedo (1997) Tan & Narasimhan (1997) Park et al. (2000)

Figure 3.2 – Sequence-Dependent Setup Scheduling Literature
for the Tardiness problem

A variety of heuristic methods, including genetic algorithms, neural nets, simulated annealing, tabu search, and other methods, have been applied to the single and multiple machine sequence-dependent tardiness problems, as shown in Figure 3.2. The heuristic literature is pointed out to show where the focus of the research has been, but tardiness heuristics are not addressed in this research and, therefore, will not be discussed in detail.

3.3 Non Sequence-Dependent Literature

Some related non sequence-dependent literature is now reviewed as background for the more current tardiness and ET literature. This earlier research is shown to better illustrate the research progression and address some of the problem characteristics of this research.

A review of the literature shows few published ET articles address scheduling jobs on multiple machines to minimize total earliness and tardiness costs. Cheng and Gupta (1989) and Baker and Scudder (1990) provide surveys on parallel machine scheduling with earliness and tardiness penalties. Baker and Scudder show that most of the models incorporating earliness and tardiness penalties are single machine models involving a common due date for all jobs. Garey et al. (1988) proved that the one-processor scheduling problem with symmetric earliness and tardiness penalties is NP-complete even without sequence-dependent setup times. This is why many studies incorporated simplifying assumptions (only one machine, all jobs ready at time zero, setup times included in processing times, no inserted idle time). Arkin and Roundy (1991) and Ghosh and Wells (1994) extended Kanet's single machine problem (Kanet 1981) and addressed the problem of scheduling N jobs on M identical parallel machines to minimize the total penalty costs for earliness and tardiness. The multi-machine problem has applications in industrial and logistical situations. A plant may have several processors able to produce final products. Scheduling on multiple machines is more complex than scheduling on a single machine, as the jobs can be split among all available machines.

Distinct due dates are a defining characteristic of both the ET and tardiness problems, and neither problem type is constrained to a zero earliness and/or tardiness solution (i.e. due dates can be missed). The assumption of distinct due dates is justified in a job-shop or make-to-order manufacturing system where job due dates are arranged between a manufacturer and the customer. Distinct due dates are particularly applicable in JIT systems where goods are produced and delivered just in time to be sold. A few

authors have included due dates as a constraint on a problem in which the objective is to minimize setup time. This constraint does not allow tardiness; therefore, a zero tardiness or zero stockout schedule must exist in order for the algorithms to work. These include Glassey (1968) and Blocher and Chand (1996) who studied single machine problems, and Uskup and Smith (1975), who studied a two-stage flowshop.

SECTION 4. MATHEMATICAL PROBLEM FORMULATIONS

AND HEURISTIC

In this section we present a new formulation and heuristic for solving ET scheduling problems with sequence dependency. The formulation is based on the traveling salesman formulation and network flow concepts. The traveling salesman (TSP) part of the formulation ensures a sequence of jobs. The network flow portion, where flow represents time, is used to 1) eliminate subtours, and 2) allow the introduction of due dates to determine tardiness.

Before presenting the scheduling model, a new basic network TSP formulation is first presented to show how the "flow portion" of the formulation works. In this formulation, the standard objective function of minimizing the total distance traveled is used. The TSP formulation is then transformed into a Due Date TSP (DDTSP), where each city is to be visited by its due date and the objective function is then one of minimizing the earliness and tardiness. The transformation of the DDTSP into a single machine ET scheduling problem with sequence-dependent setup times is then discussed. This problem will be referred to as the N/1/ET with sequence-dependent setups, as by Davis and Kanet (1993), where N refers to the number of jobs and 1 indicates one machine. Parallels between the DDTSP and scheduling problems with due dates are discussed to motivate the new formulation. Finally, the DDTSP formulation is modified to become a multiple machine scheduling model with release dates. This problem will be referred to as the N/M/ET with sequence-dependent setups and release dates, or N/M/ET-SR for short.

4.1 Basic TSP Formulation

The basic TSP formulation can be stated as a problem in which N cities must be visited by traveling a minimum distance by a single traveling salesman. The distance associated with traveling between cities i and j is denoted s_{ij}. The objective function serves to minimize the sum of the travel distances between each of the locations. The formulation of this problem is as follows.

Problem: TSP

$$Minimize \quad \sum_{i=0}^{N} \sum_{j=1}^{N} s_{ij} x_{ij} \tag{1}$$

$$Subject\ to \quad \sum_{\substack{j=0 \\ j \neq i}}^{N} x_{ij} = 1 \quad \forall i = 0...N \tag{2}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{N} x_{ij} = 1 \quad \forall j = 0...N \tag{3}$$

$$\sum_{j=1}^{N} y_{0j} = N \tag{4}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{N} y_{ij} - \sum_{\substack{i=0 \\ i \neq j}}^{N} y_{ji} = 1 \quad \forall j = 1...N \tag{5}$$

$$y_{ij} \leq N \cdot x_{ij} \quad i = 0...N,\ j = 0...N,\ i \neq j \tag{6}$$

The decision variable x_{ij} is a binary selection variable that equals 1 when city j is visited after city i and 0 otherwise. Equation (1) is the objective function that minimizes the sum of the travel distances between each of the cities. The first two constraints

(equations 2 and 3) are the arrival and departure constraints. For this problem, these constraints ensure each city i is visited only once for any selected sequence.

The next three constraints (equations 4-6) are flow constraints. The decision variable y_{ij} represents the flow from node i to node j. Equation (4) ensures the total flow out of the initial dummy/supply node of the network is equal to the total number of cities to be visited (N), the total capacity of the network. As each city is visited, the capacity is reduced by one until all cities have been visited and there is no remaining capacity. Equation (5) ensures the flow into the node minus the flow out of the node for each city (capacity consumption) is equal to one. Equation (6) ensures there is flow between two cities only if that path has been selected, and the flow (if the path is selected) is less than the total network capacity (N).

4.2 Due Date TSP Problem

The Due Date TSP is conceptually very similar to the TSP except that there is a due date for each city and thus the city should be visited by that time. The objective is no longer to minimize travel distance but to minimize the total earliness and tardiness of visiting the cities. The formulation is:

Problem: DDTSP

$$Minimize \quad \sum_{i=1}^{N} (t_i + e_i) \tag{7}$$

$$Subject\ to \quad \sum_{\substack{j=0 \\ j \neq i}}^{N} x_{ij} = 1 \quad \forall i = 0...N \tag{8}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{N} x_{ij} = 1 \quad \forall j = 0...N \tag{9}$$

$$\sum_{j=1}^{N} y_{0j} = B \tag{10}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{N} y_{ij} - \sum_{\substack{i=0 \\ i \neq j}}^{N} y_{ji} = \sum_{\substack{i=0 \\ i \neq j}}^{N} (x_{ij} \cdot s_{ij}) \quad \forall j = 1...N \tag{11}$$

$$y_{ij} \leq B \cdot x_{ij} \quad i = 0...N, \, j = 0...N, \, i \neq j \tag{12}$$

$$B - \sum_{\substack{j=0 \\ j \neq i}}^{N} y_{ij} + e_i = d_i + t_i \quad \forall i = 1...N \tag{13}$$

In this formulation, t_i represents the time past the due date (tardiness) of visiting city i and e_i represents the time before the due date (earliness) of visiting city i, so the objective function (7) is one of minimizing total earliness and tardiness. Equations (8) and (9) are the same as equations (2) and (3) of the TSP for arrival and departure constraints. The next three constraints (equations 10-12) are modified flow constraints. Constraint (10) is similar to constraint (4), except a large constant B replaces N. The flow is now a measure of the time it takes to follow the route, rather than the number of cities visited. Since the time to go from one city to another is sequence dependent, the total amount of time is unknown prior to solving the problem, thus a large constant B is needed. Equation (11) is analogous to constraint (5), except now the difference between

the flow into a city and out of that city is the amount of travel time. Equation (12) is very similar to equation (6), ensuring that flow occurs only between two cities that are connected in the route, and the network capacity is B instead of N. Finally, constraint (13) picks up the earliness and tardiness using the due date for the visit to each city.

4.3 N/1/ET with Sequence-Dependent Setups Problem

By this time, it should be readily apparent that the DDTSP is very similar to a single machine scheduling formulation with sequence dependent changeover times. Each job to be scheduled is analogous to a city that must be visited by the due date. If s_{ij} is defined as the setup time between job i to job j (instead of the travel time between city i and city j), the only changes to the DDTSP that must be made is to include the processing time of the job in constraint (11). The problem is now one of minimizing the earliness and tardiness of completed jobs, versus minimizing the earliness and tardiness of visiting cities. The equality of constraint (11) must also be changed to a 'greater than or equal to' inequality to allow the optimal insertion of idle time.

The decision variable y_{ij} continues to represent the flow from node i to node j. However, due to the introduction of job processing times in equation (11), this flow now represents the time remaining until completion of all scheduled jobs and decreases by the total of the processing and setup time of each job in the sequence. This allows us to obtain the earliness and tardiness of each job scheduled in the sequence and minimize it in the objective function.

4.4 N/M/ET with Sequence-Dependent Setups Problem

Utilizing the network structure of the DDTSP and N/1/ET with sequence-dependent setups models, it is now a simple matter to encompass the added complexity of multiple identical machines. Adding release dates to the formulation is also a simple task. DDTSP constraints (8) and (9) are slightly modified by separating out the assignment constraints into and out of the imaginary starting job 0. These assignment constraints (17) and (18) allow a number of imaginary starting jobs equal to or less than the number of machines available (M). A new constraint (23) is added to allow the addition of release dates for each of the jobs to be scheduled. With these changes and those already discussed for the N/1/ET model, the new model takes the following form:

Problem: N/M/ET with sequence-dependent setups:

$$Minimize \quad \sum_{i=1}^{N} (t_i + e_i) \tag{14}$$

$$Subject\ to \quad \sum_{\substack{j=0 \\ j \neq i}}^{N} x_{ij} = 1 \quad \forall i = 1...N \tag{15}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{N} x_{ij} = 1 \quad \forall j = 1...N \tag{16}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{N} x_{0j} \leq M \tag{17}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{N} x_{i0} \leq M \tag{18}$$

$$\sum_{j=1}^{N} y_{0j} = B \tag{19}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{N} y_{ij} - \sum_{\substack{i=0 \\ i \neq j}}^{N} y_{ji} \geq p_j + \sum_{\substack{i=0 \\ i \neq j}}^{N} (x_{ij} \cdot s_{ij}) \quad \forall j = 1...N \tag{20}$$

$$y_{ij} \leq B \cdot x_{ij} \quad i = 0...N, \ j = 0...N, \ i \neq j \tag{21}$$

$$B - \sum_{\substack{j=0 \\ j \neq i}}^{N} y_{ij} + e_i = d_i + t_i \quad \forall i = 1...N \tag{22}$$

$$B - \sum_{\substack{j=0 \\ i \neq j}}^{N} \left(y_{ij} - s_{ji}x_{ji} \right) - p_i \geq r_i \quad \forall i = 1...N \tag{23}$$

There are only n additional variables and constraints with the addition of release dates, so the model remains relatively efficient, while assuming greater capability. This constraint ensures the start time of every job is greater than or equal to the release date. The formulation of this constraint essentially subtracts the processing and setup times

from the completion time of a job to arrive at its starting point in time. This starting point is then made to be greater than or equal to the release date of the job.

Weighting factors can be added to the N/M/ET objective function (14) transforming it into a weighted ET problem objective. Neither weighting factors nor release dates were variables in this research; therefore all objective function weights are considered to be 1, and all release dates are considered to be 0.

4.5 Interpretation of Network MIP Solution

Both the x and y variables of the solution must be observed in order to construct an optimal schedule. The x variables give the order of the jobs and the order determines the setup times, but the y variables must be checked for possible inserted idle time. The y variables represent a bucket of time that remains to be used, beginning with the quantity B for the first job of the sequence. Since this bucket or quantity of time is decremented by the amount of processing, setup, and idle time (for the ET problem) for a scheduled job, idle time is simply the amount of time used beyond the processing and setup time. For example, suppose the first two jobs in a sequence are jobs 2 and 4 with 200 as the value of B. The variable y_{02} would equal 200, showing that a supply of 200 time units are available for the first assigned job. If the combined total of processing and setup time for job 2 (as the initial job in the sequence) is 11 and the value of variable y_{24} is 175, we know that there is inserted idle time of 14 time units before job 2. The original supply of 200 time units minus 175 time units remaining for the next job assignment shows 25 time units have been consumed. The 25 time units is 14 greater than the 11 required for the processing and setup time alone. Therefore, we know that job 2 starts at time 14, requires 11 time units to be completed, and is done at time 25. This can be done for each

subsequent job assignment to yield the optimal start and finish times of each job in the sequence. Tardiness problem solutions have no inserted idle time; therefore, optimal schedules can be ascertained from the x variables alone.

4.6 Zhu and Heady MIP Formulation

The Zhu and Heady MIP formulation is now presented:

$$Minimize \quad \sum_{i=1}^{N}(t_i + e_i) \tag{24}$$

$$Subject\ to \quad X_i - t_i + e_i = d_i \quad \forall i = 1...N \tag{25}$$

$$\sum_{m=1}^{M} Z_{im} = 1 \quad \forall i = 1...N \tag{26}$$

$$\sum_{\substack{j=1 \\ i \neq j}}^{N} Y_{imj} \leq Z_{im} \quad \forall i = 0...N, m = 1...M \tag{27}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{N} Y_{imj} \leq Z_{jm} \quad \forall j = 1...N, m = 1...M \tag{28}$$

$$X_i - X_j - B Y_{jim} \geq p_{im} + s_{ji} - B \tag{29}$$
$$j \neq i, \forall i = 1...N, \forall j = 0...N, m = 1...M$$

The Zhu and Heady formulation utilizes many of the same variables as the network formulation, where B is a large number; d_i is the due date for job i; t_i is the tardiness for job i, e_i is the earliness for job i, p_{im} is the processing time for job i on machine m, s_{ji} is the setup time for job i when it follows job j; N is the number of jobs to be scheduled; and M is the number of machines available. The variable X_i is the

completion time of job i. The binary variables are Y and Z, where Y_{ijm} is 1 if job i precedes job j on machine m (0 otherwise), and Z_{im} is 1 if job i is done on machine m (0 otherwise). Although the Zhu and Heady (2000) MIP model was presented with weights on the earliness and tardiness objective variables, the weights were left out as this research assumes an equal weighting of one for all problems.

The objective function (24) minimizes the sum of deviations in job completion times from the job due dates. The first constraint (25) measures the degree to which each job is tardy or early. The second constraint (26) ensures each job is processed on one and only one machine. The third and fourth constraints (27) and (28) ensure that each job comes immediately before and immediately after only one other job. The fifth constraint (29) ensures the completion time of job i is far enough after job j to include the processing time and setup time of job i (Zhu and Heady 2000).

4.7 Variable and Constraint Comparison

The network ET formulation requires $N^2 + 5N + 3$ constraints, versus $M * (N^2 + 2N + 1) + 2N$ for the Zhu and Heady formulation. It utilizes $2N^2 + 4N$ variables, $N^2 + N$ of these are binary, versus $M * (N^2 + N + 1) + 3N$, where $M * (N^2 + N + 1)$ are binary, for the Zhu and Heady formulation. It can readily be seen that although the Zhu formulation maintains a computational advantage for single machine problems, the network advantage of a smaller number of required variables and constraints increases as the number of machines grows. This is shown in Tables 4.1 and 4.2 below. The tardiness formulations would simply have n fewer variables, as the earliness variable for each job is not needed.

Formulation	# Jobs	7 Machines	9 Machines	11 Machines
Network	**10 Jobs**	153	153	153
	12 Jobs	207	207	207
	15 Jobs	303	303	303
	20 Jobs	503	503	503
Zhu	**10 Jobs**	141	383	867
	12 Jobs	193	531	1207
	15 Jobs	286	798	1822
	20 Jobs	481	1363	3127

Table 4.1 – Number of Constraints Based on Problem Size

Formulation	# Jobs	7 Machines	9 Machines	11 Machines
Network	**10 Jobs**	240	240	240
	12 Jobs	336	336	336
	15 Jobs	510	510	510
	20 Jobs	880	880	880
Zhu	**10 Jobs**	141	363	807
	12 Jobs	193	507	1135
	15 Jobs	286	768	1732
	20 Jobs	481	1323	3007

Table 4.2 – Number of Variables Based on Problem Size

4.8 Earliness/Tardiness Heuristic

A heuristic algorithm was developed and employed to efficiently solve the multiple machine ET problem. This ET heuristic utilizes some solution ideas of related problems while utilizing some new methods to exploit the structure of the problem. The ET problem is the focus of the heuristic because of the lack of research in this area, but the heuristic could likely be very effective in solving tardiness problems with only minor modifications. The algorithm consists of the following six primary steps.

Heuristic Algorithm

STEP 1: Sort the jobs to be scheduled in earliest due date (EDD) order. Jobs with common due dates will be sub-ordered by lowest processing time. Assign the jobs to machines in this order. The complexity of this step is on the order of NlogN.

STEP 2: Arbitrarily assign the first ordered job to a machine.

STEP 3: Assign the next ordered job to the machine that yields the minimum value of setup time plus tardiness (s + t) for the job being placed.

STEP 4: Upon assigning a job to a machine that has jobs already assigned, determine if any reordering should be accomplished:
- Determine if the jobs preceding the one just placed (up to 5 positions before) have any subsequent jobs with a due date within the range of the due date plus the setup and processing time (d + s + p) of the selected job. If not, skip to step 5.
- Order the job and any subsequent jobs with a due date within d + s in minimum setup order, including the setup time of one job beyond the group of jobs being reordered.

STEP 5: Return to Step 3 until all jobs are assigned to a machine. The complexity of steps 2, 3, 4, and 5 together are on the order of N.

STEP 6: Insert the optimal amount of idle time into the chosen sequences of each machine. The complexity of this step is on the order of N^2.

This algorithm utilizes all the primary characteristics of each job, including processing time, setup time, and due date to intelligently assign and order the jobs. The basic logic behind the steps of the heuristic is fairly simple – prioritize the placement of jobs by their due dates and work to minimize machine tardiness while placing the jobs. Although inserted idle time can be used to reduce or eliminate earliness in the sequence once the sequence is established, tardiness cannot be reduced, so eliminating it is a primary focus while placing the jobs. The primary parameter functions to determine were: 1) on which machine to assign a job, 2) what to use as a flag that a job should be

considered for reordering after being placed on machine, and 3) how best to reorder the jobs. A variety of heuristic parameter variations were observed into order to determine their effectiveness when developing the algorithm, but the ones utilized by the ET heuristic performed best.

Step 1 of the heuristic establishes a due date priority by assigning the jobs in due date order. This is a common and often effective ordering method in scheduling jobs with independent due dates on a single machine. In this case, the earliest due date (EDD) method establishes only an initial order from which the jobs will be assigned, and reordering may take place between job placements. A simple bubble sort was used to order the due dates, which requires NlogN calculations. The first job can be assigned arbitrarily in Step 2 as all machines are open and identical.

The combined metric of the lowest setup time plus tardiness proved most effective for machine selection. Metrics such as lowest setup time (alone), largest due date gap (with previously assigned job), and assignment in machine order were also tried. Minimizing both the setup time and the tardiness in Step 3 puts jobs on a machine where they will introduce the least additional tardiness while trying to minimize the makespan of the sequence. The job is placed on each machine and is positioned on the machine with the lowest total setup and tardiness total. As already mentioned, we attempt to minimize tardiness because once it is introduced into a sequence it can't be eliminated without changing the sequence or machine. Setup time is minimized to reduce the likelihood or magnitude of tardiness later in the sequence by reducing the sequence makespan. Although there will not always be tardiness, there will always be a setup time; therefore, this double metric penalizes more for any tardiness present, but will

always work to minimize setups and makespan. Minimizing setup time is equivalent to minimizing the sequence makespan as processing times are constant, and idle time is not inserted until the sequence is established.

Reordering of the jobs after they are individually placed on machines is an important step in reducing overall tardiness, as setup times (and subsequently makespan) are affected by each and every job placement. Once a job is placed, the ET heuristic looks back at as many as five previously scheduled jobs (depending on how many have already been scheduled) and checks for possible reordering, based on the job due dates. The most effective flag for possible job reordering in Step 4 was determined to be due dates that are close together. When the due dates of neighboring jobs (scheduled sequentially) are relatively close, it may be more advantageous to reorder them if setup time can be reduced. Another important parameter is the range of due dates (i.e. how many jobs) to consider for reordering. The chosen job's due date plus the processing and setup time (d_i to $d_i + p_i + s_i$) was found to be most effective and was used to establish a range of due dates to consider for reordering. If more than one job was included in the range of due dates, a subsequence for reordering was established. Jobs with higher processing and setup times have a larger due date range to consider and a greater chance for reordering. In order to reduce setup times and possible tardiness within the sequence on the selected machine, the jobs selected (the subsequence) are reordering based on the minimum total setup time (same as minimum makespan) of the subsequence. This proved most effective when the setup time of the first job following this subsequence was included in the calculation. This is logical as the last ordered job in the subsequence affects the setup time of the following job. Minimizing earliness and tardiness for the

subsequences was also considered as the reordering objective, but produced poorer results. Placing and reordering the jobs is done in O(N) time.

The optimal idle time insertion technique of Garey, Tarjan, and Wilfong (1988) was used with this heuristic once all jobs were placed and reordered. The basic idea behind this algorithm is jobs on a machine are grouped together (one or more groups), and each group may be moved earlier until either it can't be moved further or the number of tardy jobs is less than the number of non-tardy jobs in that block. The procedure is efficient, reasonably easy to understand and implement, and has a complexity on the order of N^2 calculations.

ET Heuristic Example

A step-by-step example of a 5 job, 2 machine problem will now be presented to illustrate the heuristic. The problem data is shown in Table 4.3 below.

Setup Times	1	2	3	4	5
0	3	10	2	7	6
1	-	5	10	8	10
2	5	-	5	8	10
3	8	2	-	1	5
4	7	4	1	-	9
5	3	5	10	8	-

Processing Times:	2	3	4	1	2
Due Dates (DD):	16	23	11	12	6
Earliest DD Order:	Job 5	Job 3	Job 4	Job 1	Job 2

Table 4.3 – ET Heuristic Example Data

Step 1: Sort the jobs to be scheduled by earliest due date (as shown above).

Step 2: Assign job 5 (the job with the earliest due date) to either machine.
- The completion time is p + s = 2 + 6 = 8.
- Since the due date is 6, the job is late 2 units.

Step 3: Determine the assignment of job 3 (the job with the second earliest due date).

- If job 3 is put on machine 1 (with job 5 already assigned), the completion time is 22 (8 + p + s = 8 + 4 + 10 = 22).
- Since the due date for job 3 is 11, setup time plus tardiness is 21 units (s + t = 10 + 11 = 21).
- If job 3 is put on machine 2 (alone), the completion time is 6 (p + s = 4 + 2 = 6), so there is no tardiness, and s + t = 2.
- Since 2 is less than 21, job 3 is assigned to machine 2.

Repeat Step 3: Determine the assignment of job 4.
- On machine 1 (with job 5), s + t = 13.
- On machine 2 (with job 3), s + t = 1.
- Since 1 is less than 13, job 4 is assigned to machine 2 after job 3.

Step 4: Since there are now 2 jobs on machine 2, check for possible reordering.
- The due date of job 4 is less than $d_3 + p_3 + s_{03} = 11 + 4 + 2 = 17$, so reordering is investigated further.
- The total setup time for the job order 3, 4 is 3 ($s_3 + s_4 = 3$) and 8 for the order 4, 3 ($s_4 + s_3 = 8$), so the order is not changed because reordering does not reduce the total setup time of the sequence.

Repeat Step 3: Determine the assignment of job 1.
- On machine 1 (with job 5), s + t = 3.
- On machine 2 (with jobs 3 and 4), s + t = 8.
- Therefore, job 1 is assigned to machine 1 after job 5.

Step 4: Since there are now 2 jobs on machine 1, we must check for possible reordering.
- Since the due date of job 1 is greater than the due date of job 5 plus the processing and setup time of job 5, no reordering is done.

Repeat Step 3: Determine the assignment of job 2.
- On machine 1 (with job 5 and 1), s + t = 5
- On machine 2 (with jobs 3 and 4), s + t = 4
- Therefore, job 2 is assigned to machine 2 after job 4

Step 4: Since there are more than 2 jobs on machine 1, check for possible reordering.
- Since the due date of job 2 is greater than the due date of job 4 plus the processing and setup time of job 4, no reordering is done and sequencing is complete.

Step 6: Insert the optimal amount of idle time.
- The final sequences are: Machine 1: 5, 1 and Machine 2: 3, 4, 2.
- For machine 1, job 5 is started at time zero and 3 time units of idle time are inserted before job 1.
- For machine 2, job 3 is started at time 5 and 3 time units are inserted before job 2.
- The ET objective function for both machines is 3 and this is optimal.

In addition to the ET heuristic just described, a simple earliest due date (EDD) 'shuffle' heuristic algorithm was developed as a baseline for comparison. In this heuristic the jobs are sorted and placed by due date, as in the ET heuristic, but the algorithm is completed by simply assigning the jobs in machine order. For example, if the jobs sorted by due date are in the order 4, 2, 3, 1, 5, and 2 machines are being used, job 4 would be assigned to machine 1, job 2 would be assigned to machine 2, job 3 would be assigned to machine 1, and so forth. Jobs 4, 3 and 5 would be assigned to machine 1, and jobs 2 and 1 would be assigned to machine 2.

SECTION 5. RESEARCH AND EXPERIMENTAL DESIGN

5.1 Research Goal

The general research goal explored is: Determine an efficient and effective way to solve the generalized multi-machine, sequence dependent setup scheduling problem with tardiness and earliness/tardiness (ET) objectives.

5.2 Research Objectives

The general research objectives are:

1) Examine the capabilities of a new mixed-integer programming (MIP) formulation for the multi-machine, sequence dependent setup, scheduling problem with both tardiness and ET objectives.

   a) Compare the new MIP formulation to the Zhu and Heady (2000) formulation in terms of solution time and problem size for the tardiness problem.

   b) Compare the new MIP formulation to the Zhu and Heady (2000) formulation in terms of solution time and problem size for the ET problem.

2) Examine the capabilities of an ET heuristic for the multi-machine, sequence dependent setup, ET scheduling problem.

   a) Compare the heuristic solutions with the optimal solutions of the new MIP formulation to determine the efficacy and accuracy of the model.

   b) Examine the capability of the heuristic in terms of problem size.

5.3 Research Questions

The research questions are:

1. Which model is more efficient in terms of average solution time, the Zhu and Heady (2000) formulation or the network-based formulation, presented here, for the tardiness problem?

2. Which model is more efficient in terms of average solution time, the Zhu and Heady (2000) formulation or the network-based formulation for the ET problem?

3. How does the number of jobs and number of machines affect the solution time for both MIP formulations for the tardiness problem?

4. How does the number of jobs and number of machines affect the solution time for both MIP formulations for the ET problem?

5. How accurate, in terms of deviation from optimal (using MIP solutions for comparison), is the ET heuristic for the ET problem?

6. What is the order of complexity of the ET heuristic and can it efficiently solve larger scale problems than the MIP models can solve?

The two solution methods to be employed are: (1) Mixed-integer optimization and (2) an earliness/tardiness heuristic. These methods are described below.

5.4 Mixed-Integer Optimization

CPLEX version 7.1 was used to solve the mixed-integer formulations and was run on a one giga-hertz personal computer. CPLEX is a commercially available software package that efficiently solves mixed-integer formulations. The CPLEX Mixed Integer

Optimizer exploits a branch and cut algorithm to solve a series of Linear Programming (LP) sub-problems. To manage the sub-problems efficiently, CPLEX builds a tree in which each sub-problem is a node. The root of the tree is the LP relaxation of the original MIP problem. CPLEX default settings were used in solving all mixed integer problems; therefore, all cuts were selected automatically and the "best bound" variable selection strategy was utilized. The best bound strategy concentrates on exploring nodes that are high in the branch and cut tree for the purpose of proving optimality more quickly.

Cuts are constraints added to the model to cut away non-integer solutions that would otherwise be solutions of the LP relaxation. The addition of cuts usually reduces the number of branches needed to solve a MIP. The CPLEX program utilizes nine types of cuts known as Clique, Cover, Disjunctive, Flow Cover, Flow Path, Gomory Fractional, Generalized Upper Bound (GUB), Implied Bound, and Mixed Integer Rounding (MIR) cuts. Clique cuts exploit a relationship among a group of binary variables such that at most one variable in the group can be positive in any integer feasible solution. If a constraint takes the form of knapsack constraint, then there is a minimal cover associated with the constraint and CPLEX can generate a constraint corresponding to this condition. A MIP problem can be divided into two subproblems with disjunctive feasible regions of their LP relaxations by branching on an integer variable. Disjunctive cuts are inequalities valid for the feasible regions of LP relaxations of the subproblems, but not valid for the feasible region of LP relaxation of the MIP problem. Flow covers are generated from constraints that contain continuous variables, where the continuous variables have variable upper bounds that are zero or positive depending on the setting of associated

binary variables. Flow path cuts are generated by considering a set of constraints containing the continuous variables that describe a path structure in a network, where the constraints are nodes and the continuous variables are in-flows and out-flows. Gomory fractional cuts are generated by applying integer rounding on a pivot row in the optimal LP tableau for a basic integer variable with a fractional solution value. A GUB constraint for a set of binary variables is a sum of variables less than or equal to one. If the variables in a GUB constraint are also members of a knapsack constraint, the minimal cover can be selected considering at most one of the GUB constraint members can be one in the solution. In some models, binary variables imply bounds on continuous variables. CPLEX generates potential cuts to reflect these relationships. MIR cuts are generated by applying integer rounding on the coefficients of the integer variables and the right-hand side of a constraint (ILOG, 2001).

The presented network MIP formulation was compared to the Zhu and Heady formulation in solving the N/M/T and N/M/ET problems, both with sequence-dependent setups. The software package CPLEX 7.1 was used to solve five problems (replications) for most cells of four different job sizes (10, 12, 15, and 20 jobs) and three quantities of machines (1, 3, and 7 machines), as shown in Table 5.4. Twenty-five problems were solved for the cells that represent the lowest computational complexity (10 jobs with 1 and 3 machines) so statistical testing of these cells might be accomplished. Based on pilot study results, these experimental cells are the most likely to be solved to completion, but great variation in solutions times will require a higher sample size. Ten problems were solved for all 12 job problems in order to add additional data for trend analysis. All other variables were held constant. This resulted in an experimental design with 2

objectives, 2 formulations, 4 job quantities, 3 machine quantities, and 5, 10 or 25 replications for a total of 460 problems.

Objective	Formulation	# Jobs	1 Machine	3 Machines	7 Machines
Tardiness	Network	10 Jobs	25 problems	25 problems	5 problems
		12 Jobs	10 problems	10 problems	10 problems
		15 Jobs	5 problems	5 problems	5 problems
		20 Jobs	5 problems	5 problems	5 problems
	Zhu and Heady	10 Jobs	25 problems	25 problems	5 problems
		12 Jobs	10 problems	10 problems	10 problems
		15 Jobs	5 problems	5 problems	5 problems
		20 Jobs	5 problems	5 problems	5 problems
ET	Network	10 Jobs	25 problems	25 problems	5 problems
		12 Jobs	10 problems	10 problems	10 problems
		15 Jobs	5 problems	5 problems	5 problems
		20 Jobs	5 problems	5 problems	5 problems
	Zhu and Heady	10 Jobs	25 problems	25 problems	5 problems
		12 Jobs	10 problems	10 problems	10 problems
		15 Jobs	5 problems	5 problems	5 problems
		20 Jobs	5 problems	5 problems	5 problems

Table 5.4 – Experimental Design for MIP Model Comparison

For this research, a reasonable amount of time (the allotted time) is defined as a maximum solution time of 7,200 seconds or 2 hours for each individual problem.

Job processing times were randomly selected from a discrete uniform distribution with a range of 1 to 10 time units. The setup times for the test problems were also generated randomly from a discrete uniform distribution with a range of 1 to 10 time units. Due dates for the problems were randomly generated from a discrete uniform distribution with a range determined by the number of jobs and machines in order to assure the problems are not trivial.

The due date range for each problem cell was obtained by multiplying the number of jobs by the expected average value of 11 time units for each job (processing time plus

setup time) and dividing this by the number of available machines. This yields a maximum value of the range from which to obtain a random due date.

The value of B used for all MIP problems was held constant in order to obtain consistent solution time results. A value of 300 was used in order to be large enough for all problems solved in the experimental design. This value was based on the 20 job 1 machine problem cell, as it has the potential to use the greatest amount of time units, of which B represents the initial supply. Since each job should average a total of 11 processing and setup time units, a B value of 220 (11 multiplied by 20 jobs) should be sufficient for most problems. The value of 300 was used in order to be conservative, due to the random generation of the problems and to ensure B is large enough for any inserted idle time. Each problem solution was checked to ensure the initial bucket of time, represented by B, did not run out before all jobs of the solution sequence were processed.

The generation of these parameters is consistent with previous research in this area. The programs, coded in C language, used to generate the MIP problems solved with CPLEX are listed in appendices A, B, C, and D.

5.5 ET Heuristic

The ET and EDD heuristics were developed/coded using the programming language C. The resultant programs were run on the same one giga-hertz personal computer as the MIP formulations in order to be comparable to the results (in terms of solution time/computing time) of those formulations.

The developed ET heuristic was compared with the best solutions of the MIP formulations for 10, 12, 15, and 20 job problems in order to determine its effectiveness. It was then run for 40 and 80 job problems and compared with the EDD heuristic

algorithm. This was done to determine how well it performs for smaller scale problems solvable by the MIP models, and larger problems beyond the capability of these models. The heuristic runs on the order of $O(N^2)$ time, so it is very fast. This run time is based on the inserted idle time algorithm running in $O(N^2)$ time for single machines. When the assigned jobs are split among multiple machines, run time is markedly reduced, as multiple smaller numbers of jobs per machine are much faster to run. Actual run times will vary greatly depending on each problem's characteristics and the number of subsequences checked for reordering. The experimental design is shown in Table 5.5.

Method	Problem Size	1 Machine	3 Machines	7 Machines
EDD Heuristic	40 Jobs	5 problems	5 problems	5 problems
	80 Jobs	5 problems	5 problems	5 problems
ET Heuristic	10 Jobs	5 problems	5 problems	5 problems
	12 Jobs	5 problems	5 problems	5 problems
	15 Jobs	5 problems	5 problems	5 problems
	20 Jobs	5 problems	5 problems	5 problems
	40 Jobs	5 problems	5 problems	5 problems
	80 Jobs	5 problems	5 problems	5 problems

Table 5.5 – Experimental Design for Heuristic Comparison

Three quantities of machines (1, 3, and 7) were used in the comparison, and each cell consists of 5 replications. This is done to assess the heuristic's efficiency and adds 120 problems to the experimental design. The heuristic programs, coded in C language, are listed in appendices E and F.

SECTION 6. EXPERIMENTAL RESULTS

The network-based MIP formulation solved all multiple machine problems faster on average than the Zhu and Heady MIP formulation, whereas the Zhu and Heady formulation solved the ET single machine problems faster. The developed heuristic yielded good results that took less than a second to produce for every problem run.

6.1 MIP Formulation – Earliness/Tardiness Problem

The network-based MIP formulation enabled the solution of larger problems, as shown in Table 6.6 below. Experimental results show the network formulation to be more efficient for multi-machine problems, while the Zhu and Heady formulation was more efficient for single machine problems. Only for 15 jobs and 1 machine did the Zhu and Heady MIP formulation solve more problems to completion, but even for this case it only solved 2 problems, versus 1 for the network formulation. The network formulation solved all 10 job problems, most of the 12 job problems, and some of the 15 job problems. Neither formulation could solve any of the 20 job problems to completion.

Problem	Formulation	Problem Size	1 Machine	3 Machines	7 Machines
ET	Network	10 Jobs	100%	100%	100%
		12 Jobs	90%	100%	100%
		15 Jobs	20%	20%	80%
		20 Jobs	0%	0%	0%
	Zhu and Heady	10 Jobs	100%	84%	40%
		12 Jobs	90%	50%	0%
		15 Jobs	40%	0%	0%
		20 Jobs	0%	0%	0%

Table 6.6 – Percentage of ET Problems Solved

The network formulation solved all multi-machine problems faster on average, when the problems were solved to completion, than the Zhu and Heady model. The Zhu and Heady model solved the single machine problems faster on average. The gap between average solution times widened, as shown in Figures 6.2 and 6.3 below, for multi-machine problems as the number of machines increased. This was due to the Zhu and Heady formulation solution times increasing as the number of machines increased, while the network solution times decreasing as the number of machines increased. The network formulation problem structure enabled the CPLEX MIP software to solve problems with a greater number of machines increasingly faster by enabling a greater number of flowcuts, thereby reducing the feasible solution region much faster.
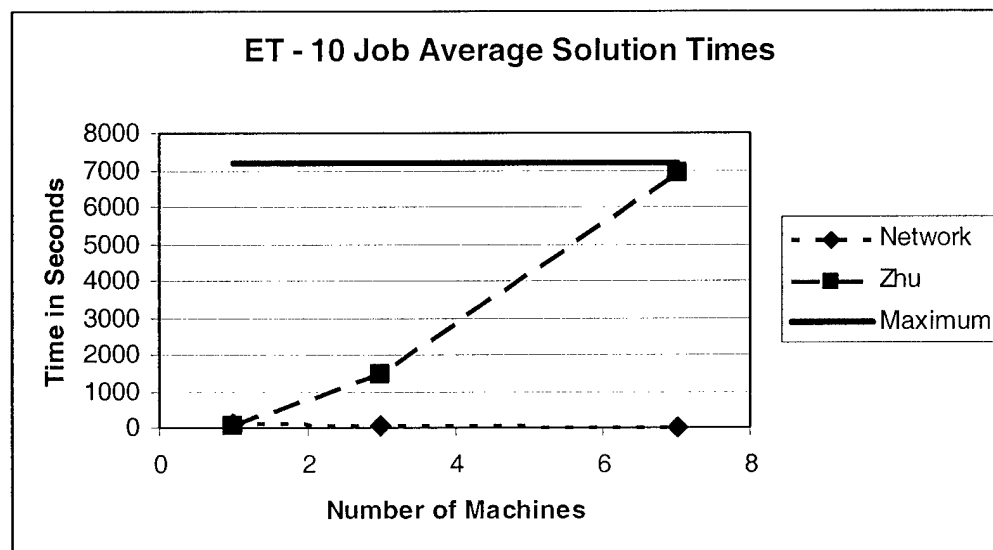
Figure 6.2 – Average 10 Job Solution times for the ET problem
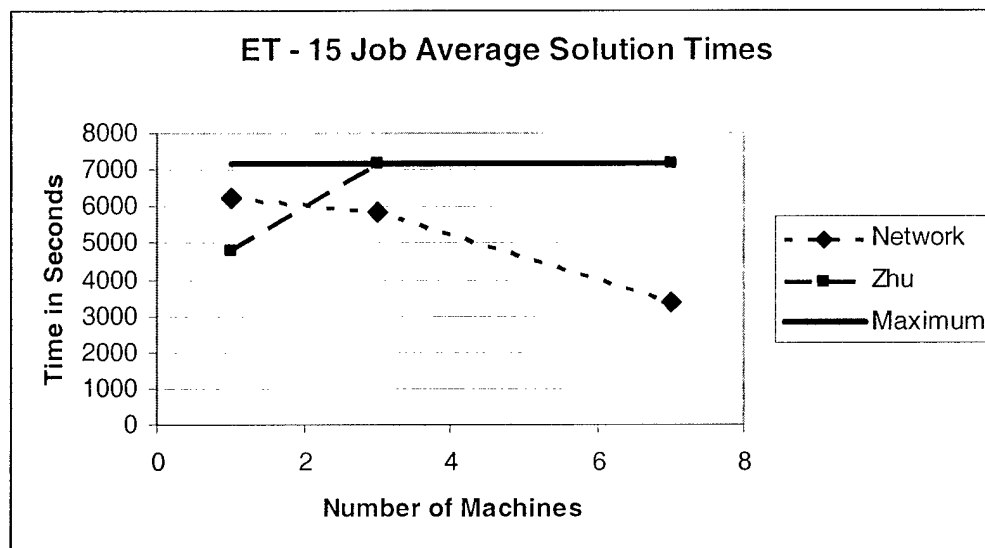
Figure 6.3 – Average 15 Job Solution times for the ET problem

The run/solution times of problems that were stopped at the maximum of 7200 seconds (2 hours) were included in the average. Therefore, cells with average solution times that approach 7200 seconds consist of a greater number of problems that did not solve to completion. Even with this truncated result for the Zhu and Heady average solution time, the network formulation solved the 10 job, 7 machine problems an average of 3,448 times faster.

Problem	Formulation	Problem Size	1 Machine	3 Machines	7 Machines
ET	Network	10 Jobs	110	58	2
		12 Jobs	2330	1834	120
		15 Jobs	6250	5841	3366
		20 Jobs	7200	7200	7200
	Zhu and Heady	10 Jobs	40	1449	6896
		12 Jobs	854	3927	7200
		15 Jobs	4789	7200	7200
		20 Jobs	7200	7200	7200

Table 6.7 – Average ET Run Times in Seconds

The network formulation also performed much better for the multi-machine cells where neither formulation solved the problems to completion, as shown in Table 6.8. For example, the network formulation solutions were an average of 63 percent lower than the Zhu and Heady formulation solutions for the 20 job and 7 machine problems. The Zhu and Heady formulation maintained a solution advantage for the single machine problems, which increased as the number of jobs increased. Gap percentages were calculated using the formula (NO − ZO) / maximum(NO,ZO) * 100, where NO is the network objective function value and ZO is the Zhu and Heady objective function value.

Problem	Formulation	Problem Size	1 Machine	3 Machines	7 Machines
ET	Network	10 Jobs	0%	0%	0%
		12 Jobs	0%	0%	-8%
		15 Jobs	7%	-3%	-47%
		20 Jobs	31%	-53%	-63%

Table 6.8 – Solution Gap Percentages Between ET Formulations

Although 25 problems were selected for the 10 job, 1 machine and 10 job, 3 machine cells in order to facilitate statistical testing, such testing did not prove useful, based on the results. All 25 generated problems were solved to completion for only the 10 job and 1 machine ET experimental cell. Since statistical testing of one cell would show very little, no statistical tests were accomplished. The difference between the methods is more evident with the increase in the number of machines, but the data points can't easily be obtained or compared, since solution times are limited to 2 hours. The results show an evident computational advantage for the network formulation when solving multiple machine problems, and a computational advantage for the Zhu and Heady formulation when solving single machine problems.

6.2 MIP Formulation – Tardiness Problem

As with the ET problems, the network-based MIP formulation enabled the solution of larger problems, as shown in Table 6.9 below. Experimental results once again show the network formulation to be more efficient for multi-machine problems. However, the Zhu and Heady formulation showed no obvious computational advantage for single machine problems for tardiness problems. The network formulation solved all 10 job problems and an increasing number of multi-machine problems as the number of machines increased. The Zhu and Heady formulation could not solve all the problems for any experimental cell, and it solved more problems than the network formulation only for the 15 job, 1 machine cell. Again, neither formulation could solve any of the 20 job problems to completion.

Problem	Formulation	Problem Size	1 Machine	3 Machines	7 Machines
Tardiness	Network	10 Jobs	100%	100%	100%
		12 Jobs	40%	60%	100%
		15 Jobs	0%	20%	80%
		20 Jobs	0%	0%	0%
	Zhu and Heady	10 Jobs	84%	32%	0%
		12 Jobs	40%	0%	0%
		15 Jobs	20%	0%	0%
		20 Jobs	0%	0%	0%

Table 6.9 – Percentage of Tardiness Problems Solved

The network formulation solved all multi-machine problems faster on average than the Zhu and Heady model, when the problems were solved to completion, as shown in Table 6.10. The Zhu and Heady model solved some 15 job single machine problems, and therefore had a slightly faster average solution time. However, the network formulation solved the 10 job single machine problems notably faster than the Zhu and Heady formulation. This is a reversal from the ET formulation results. As with the ET

problem results, the gap between average solution times widened greatly for multi-machine problems as the number of machines increased. This was again due to the Zhu and Heady formulation solution times increasing as the number of machines increased, while the network solution times decreasing as the number of machines increased. Figures 6.4 and 6.5 below clearly show the widening gap in solution times, as already seen for the ET problem.
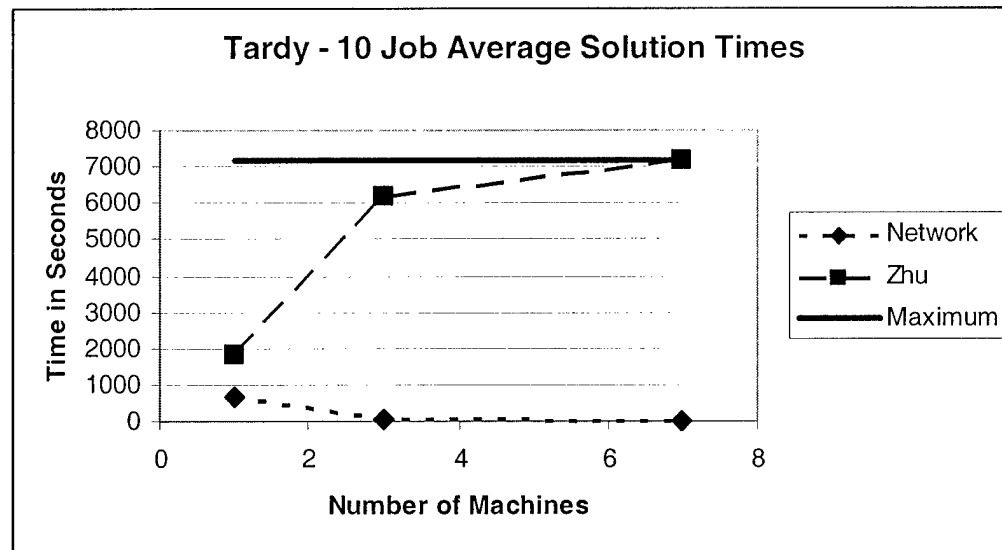
Figure 6.4 – Average 10 Job Solution times for the Tardiness problem

Tardy - 15 Job Average Solution Times
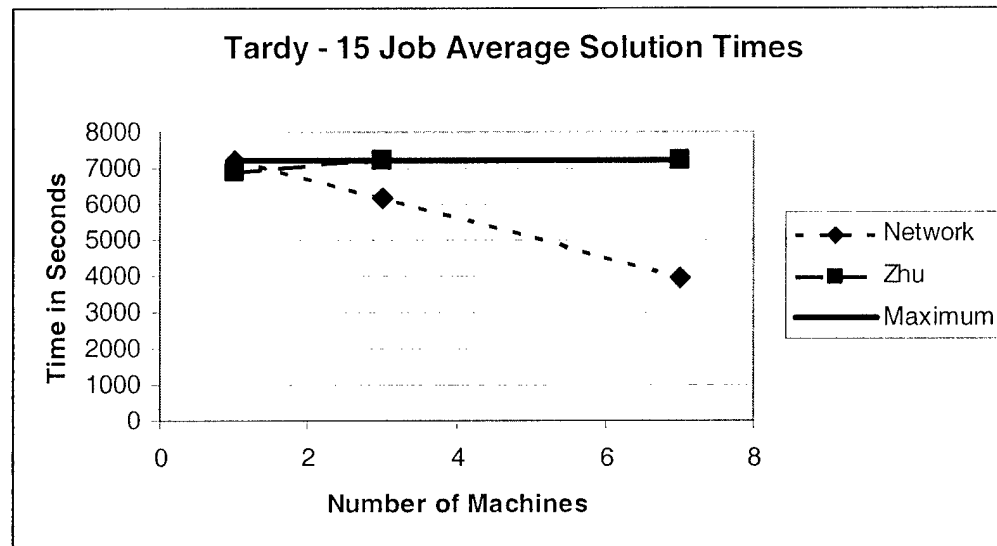
Figure 6.5 – Average 15 Job Solution times for the Tardiness problem

The average 10 job, 7 machine solution time is 7200 seconds because no problems were solved in the maximum 7200 second allotted solution time. Even though the average solution times would have been greater than 7200 seconds for the Zhu and Heady formulation, the network formulation still solved the 10 job, 7 machine problems an average of 3,600 times faster.

Problem	Formulation	Problem Size	1 Machine	3 Machines	7 Machines
Tardiness	Network	10 Jobs	679	80	2
		12 Jobs	4349	4223	116
		15 Jobs	7200	6186	3955
		20 Jobs	7200	7200	7200
	Zhu and Heady	10 Jobs	1800	6190	7200
		12 Jobs	4577	7200	7200
		15 Jobs	6869	7200	7200
		20 Jobs	7200	7200	7200

Table 6.10 – Average Tardiness Solution Times in Seconds

The network formulation obtained better results (lower objective function) for every cell, except 15 jobs and 1 machine, where the Zhu and Heady formulation results

were an average of 11 percent lower. As with the ET problem, the network formulation also performed much better for the multi-machine cells where neither formulation solved the problems to completion, as shown in Table 6.11. The network formulation solutions achieved an average 85 percent lower solution (gap) than the Zhu and Heady formulation solutions for the 20 job and 7 machine problems.

Problem	Formulation	Problem Size	1 Machine	3 Machines	7 Machines
Tardiness	Network	10 Jobs	-2%	-9%	-6%
		12 Jobs	-6%	-45%	-42%
		15 Jobs	11%	-64%	-59%
		20 Jobs	-55%	-63%	-85%

Table 6.11 – Solution Gap Percentages Between Tardiness Formulations

No statistical testing was accomplished for the tardiness problems, as the Zhu and Heady formulation did not solve all of the problems to completion for any cell of the experimental design. However, the difference between the methods is even more evident for the tardiness problems than with the ET problems. The experimental results show an evident and increasing computational advantage for the network formulation as the number of machines is increased for multiple machine problems. A computational advantage was not evident for all single machine problems, as each formulation performed better with different job sizes. However, there is more evidence showing the network formulation was more efficient for single machine problems. The average 15 job advantage for the Zhu and Heady solutions was small, whereas the average network advantage for 20 jobs was more pronounced. The network formulation also produced better average solutions for both the 10 and 12 job cells.

6.3 Earliness/Tardiness Heuristic

The ET heuristic yielded good solutions in an insignificant amount of computation time. All solutions were obtained in less than one second. Although only two optimal solutions were noted for the ET heuristic, the gap between the heuristic solutions and the best solution yielded by both MIP formulations are relatively small. The gaps can be considered relatively small, as having a single job out of optimal position early in a sequence can cause the objective function to be 2 or 3 times larger than the optimal sequence for ET problems. Additional optimal solutions may have been obtained as better solutions were obtained by the heuristic for 4 of the 15 problems with 20 jobs. This was possible because the MIP formulations could not solve the 20 job problems to completion, and were stopped before optimal solutions were reached. As shown in Table 6.12, the best MIP solution was between an average of 2 and 38 percent less than the heuristic solution. All of the 10 and 12 job MIP solutions are optimal and some of the 15 job solutions are optimal, but none of the 20 job problems were solved to completion.

Problem	Method	Problem Size	1 Machine	3 Machines	7 Machines
ET	ET Heuristic	10 Jobs	13%	27%	11%
		12 Jobs	28%	31%	18%
		15 Jobs	22%	38%	19%
		20 Jobs	2%	17%	25%

Table 6.12 – ET Solution Gap Percentages Between Heuristic and Best MIP Solution

The developed heuristic performed very well when compared with a simple algorithm that assigns a job to each machine one at a time in earliest due date (EDD) order. All heuristic solutions were again obtained in less than 1 second and were consistently better than the EDD solutions. The ET heuristic solutions were between 24

and 61 percent better (lower) on average than the EDD heuristic for the 40 and 80 job problems (see table 6.13). These problems are far beyond the capabilities of the MIP models to solve to completion.

Problem	Method	Problem Size	1 Machine	3 Machines	7 Machines
ET	ET Heuristic	40 Jobs	-55	-24	-52
		80 Jobs	-61	-48	-60

Table 6.13 – ET Solution Gap Percentages Between Heuristic and EDD Method

The metrics and objectives utilized within the heuristic worked well in conjunction with each other and produced a very capable algorithm. This heuristic provides a very fast and efficient alternative with which to solve problems too large and computationally complex for any current MIP formulation.

6.4 Other Results

The impact of the network formulation constant B size (a sufficiently large value) on the solution times of tardiness problems was also investigated. B represents an initial supply or bucket of time of which job processing and setup times consume. As a job is added to the schedule, the supply of time is reduced; therefore, B must be large enough to supply the processing and setup time for every job of the optimal sequence or an optimal solution can't be obtained. Determining a minimum value for B is not difficult, as it can simply be made large enough to enable completion of the longest possible sequence, but finding a good value for B that reduces problem run time is difficult. It was found that solution times for the same problem varied greatly with changes in the value of B and those changes were highly variable for different problems.

Changes in the value of B (network capacity) used in the problem formulations were explored for five tardiness problems of ten jobs. The baseline value of B was

calculated by summing all job processing times and the largest changeover time for each job. This represents the smallest value of B that ensures optimality. The same problem was then solved using increments of B minus ten until the search space became infeasible or a non-optimal solution was obtained. The last increment of B used to solve each problem was based on the smallest possible value, which is the makespan of the optimal sequence. In Figure 6.6 below, the x-axis is the value of B used and the y-axis is the solution time of the problem. The figure shows the smallest solution time was always obtained using the smallest possible B, but other values of B yielded highly variable solution times. Variations in the value of B changed the feasible search region and problem structure in a way that resulted in great time fluctuations.
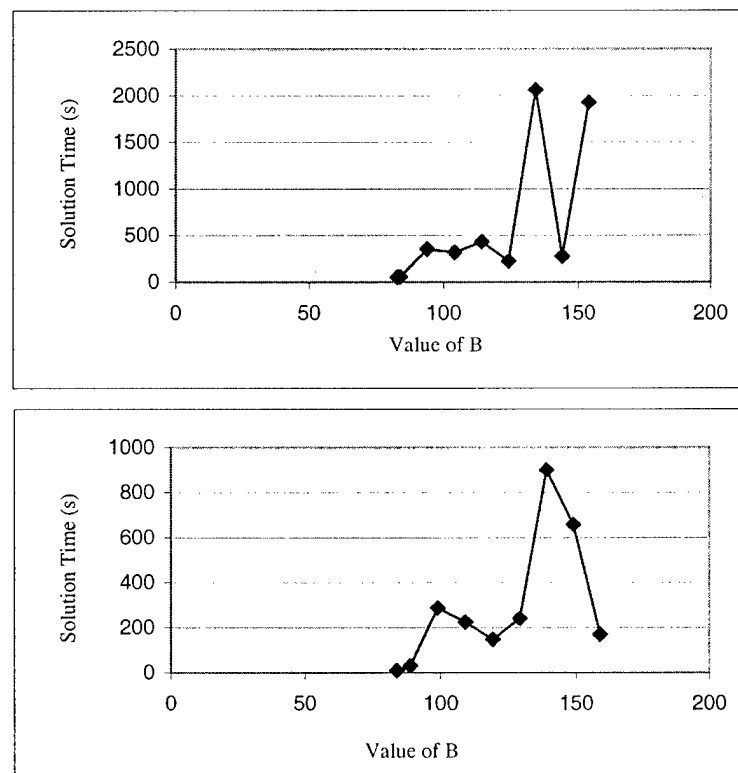
Figure 6.6 – Solution Times for Changes in B

The only conclusion that could be drawn was that the problem solution times were generally very fast if B was either exactly the size of the problem's optimal solution

makespan or very close. This observation presented the problem of how to find the makespan of a problem for which there is not yet a solution. An evolutionary algorithm was used to find a good feasible solution from which to obtain a makespan value and value for B. This tactic proved very effective in most cases and solution times for the combination of the evolutionary algorithm and the MIP together were much faster than the MIP alone. However, this method introduced the opportunity for non-optimal solutions. Some instances arose where the makespan value used for B, determined by the evolutionary algorithm, was less than the makespan needed for the optimal solution. In this case the problem solution is constrained to complete within the smaller makespan and optimality can't be reached using the MIP formulation.

An evolutionary algorithm was also utilized in determining a good upper bound. The best (lowest) solution obtained with the algorithm was entered in CPLEX as the upper bound, thereby decreasing the feasible search space. However, this did not significantly improve solution times.

SECTION 7. CONCLUSIONS

7.1 Conclusions

The presented MIP formulation provides a unique and useful method of conceptualizing and modeling a practical, yet difficult, problem within industry. Although a few solution methods have been applied to this problem, this new network-based MIP formulation adds a greater level of structure to the N/M/T and N/M/ET problems with sequence dependent setups and gives added capability in solving larger multi-machine problems. This research shows the new MIP model is much more efficient in terms of computation time for multi-machine problems than the Zhu and Heady generalized formulation of these problems. The structure of the model, which adapted a network-based TSP structure to multiple machines, enabled the CPLEX MIP software to solve problems with a greater number of machines increasingly faster as machines were added. The mixed-integer nature of the formulation allows the solution of this class of problems by companies with any one of a number of commonly available integer programming software packages. The ET heuristic provides another effective tool in solving larger problems of this class where the MIP formulations become computationally too difficult to solve in a reasonable amount of time.

7.2 Limitations

This research was necessarily limited in scope primarily due to the large computation times associated with these problems. The variables of number of jobs, number of machines, and replications per cell were limited to a relatively small number

in order to complete the experimental design in a reasonable amount of time. This research shows distinct trends in solution run times based on the number of jobs and machines, but the results are limited by the experimental design. Variables for weighted earliness and tardiness penalties and ready times/dates for jobs were held constant.

7.3 Future Research

As already discussed, there are many useful extensions to the network MIP model research. Greater computational experience would be very interesting and useful, especially if it involved ready times and/or weighted earliness and tardiness penalties. Improvements to the network formulation that eliminate variables or reduce the feasible search region could improve the computational speed of the model. The determination of good bounds may also improve the computational speed of the model, as previous research on tardiness and ET sequence dependent setup problems has failed to produce a tight lower bound.

The heuristic worked well, but there is room for improvement. A more complex scheme for simultaneous and dynamically placing and reordering the jobs on the machines could be more effective. Any one of the many hill-climbing algorithms, such as tabu search, simulated annealing, or genetic search could also be added to the heuristic to make it a more effective meta-heuristic. Another approach would be to use the optimal inserted idle time algorithm at every placement or reordering of a job versus only after the sequence is established. This would have to be done carefully as the inserted idle time dynamics of a problem can change greatly with the addition of each additional job.

APPENDIX A – NETWORK ET PROBLEM BUILDING C CODE

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include <time.h>

#define RANGE1 10
#define RANGE2 10
#define seed 10


/**********************************************************************

THIS PROGRAM OUTPUTS MATHEMATICAL NETWORK PROGRAMMING FORMULATIONS FOR
EARLINESS/TARDINESS SCHEDULING PROBLEMS WITH SEQUENCE DEPENDENT SETUPS,
BASED ON AN INPUT NUMBER OF JOBS, MACHINES, DUE DATE RANGE, AND NUMBER
OF PROBLEMS TO BE CREATED.

**********************************************************************

Variables:
n - Number of jobs (input)
np - Number of problems to be output (input)
i,j,k - Loop indexes
x - Counter
s - Setup times
p - Processing times
d - Due dates/times
B - A sufficiently large number
mc - Maximum changeover time
m - Number of machines
a - Temporary value for B - d (large number minus due date)
ddr - Maximum value of due date range
RANGE1 - Maximum of the range of random processing times
RANGE2 - Maximum of the range of random changeover times
*/

int rnd(int range);
void seedrnd(void);

void main()

{
    int n,np,i,j,k,x,s[25][25],p[25],d[25],B,mc,m,a,ddr;

    seedrnd();

/*Input the number of jobs and problems*/

    cout <<"Enter the number of jobs:";
    cin >>n;
```

```
cout <<"Enter the number of machines:";
cin >>m;

cout <<"Enter the due date range:";
cin >>ddr;

cout <<"Enter the number of problems:";
cin >>np;

for (k=1;k<=np;k++)
```

/*This section creates an output file name based on the number of jobs
and an index number*/

```
{
        char name[20]="a:1atmm";
        char buffer1[5];
        char buffer2[5];
        char buffer3[5];
        char ext[5]=".lp";

        itoa (n,buffer1,10);
        strcat(name,buffer1);
        itoa (m,buffer3,10);
        strcat(name,buffer3);
        itoa (k,buffer2,10);
        strcat(name,buffer2);
        strcat(name,ext);

        fstream f;
        f.open(name,ios::out);
```

/*Generates processing times, setup times, due dates, and B*/

```
        B=0;
        for (i=1;i<=n;i++)
        {
                p[i]=rnd(RANGE1)+1;
                B=B+p[i];
                d[i]=rnd(ddr)+1;
                mc=0;
                for (j=0;j<=n;j++)
                {
                        if (i!=j)
                        {
                                s[j][i]=rnd(RANGE2)+1;
                                if (s[j][i]>mc)
                                {
                                        mc=s[j][i];
                                }
                        }
                }
                B=B+mc;
        }
        B=300;
```

/*Prints constants*

```
f <<"processing times: ";
for (i=1;i<=n;i++)
{
        f <<p[i]<<"   ";
}
f <<endl;
f <<"due dates: ";
for (i=1;i<=n;i++)
{
        f <<d[i]<<"   ";
}
f <<endl;
f <<"setup times:"<<endl;
for (i=0;i<=n;i++)
{
        for (j=1;j<=n;j++)
        {
                if (i!=j)
                {
                        f <<s[i][j]<<"   ";
                }
                else
                {
                        f <<"      ";
                }
        }
        f <<endl;
}


/*This section prints the objective function*/

        f <<"Minimize"<<endl;
        f <<"T1 + U1";
        for (i=2;i<=n;i++)
        {
                f <<" + T"<<i<<" + U"<<i;
        }
        f <<endl;

/*This section prints constraint 1*/

        f <<"Subject To"<<endl;
        for (i=1;i<=n;i++)
        {
                x=0;
                for (j=0;j<=n;j++)
                {
                        if (j!=i)
                        {
                                if (x>0)
                                {
                                        f <<" + ";
                                }
                                f <<"x"<<i<<"x"<<j;
                                x=x++;
                        }
                }
```

```
            }
    f <<" = 1"<<endl;
    }


/*This section prints constraint 2*/

    for (j=1;j<=n;j++)
    {
        x=0;
        for (i=0;i<=n;i++)
        {
            if (i!=j)
            {
                if (x>0)
                {
                    f <<" + ";
                }
                f <<"x"<<i<<"x"<<j;
                x=x++;
            }
        }
    f <<" = 1"<<endl;
    }


/*This section prints constraint 3*/

    f <<"x0x1";
    for (j=2;j<=n;j++)
    {
        f <<" + x0x"<<j;
    }
    f <<" <= "<<m<<endl;


/*This section prints constraint 4*/

    f <<"x1x0";
    for (j=2;j<=n;j++)
    {
        f <<" + x"<<j<<"x0";
    }
    f <<" <= "<<m<<endl;


/*This section prints constraint 5*/

    for (j=1;j<=n;j++)
    {
        x=0;
        for (i=0;i<=n;i++)
        {
            if (i!=j)
            {
                if (x>0)
                {
                    f <<" + ";
                }
                f <<"y"<<i<<"y"<<j<<" - y"<<j<<"y"<<i;
                x=x++;
```

```
                    }
            }
            x=0;
            for (i=0;i<=n;i++)
            {
                    if (i!=j)
                    {
                            f <<" - ";
                            f <<s[i][j]<<" x"<<i<<"x"<<j;
                    }
            }
            f <<" >= "<<p[j]<<endl;
        }

    /*This section prints constraint 6*/

        for (i=0;i<=n;i++)
        {
                for (j=0;j<=n;j++)
                {
                        if (j!=i)
                        {
                                f <<B<<" x"<<i<<"x"<<j<<" -
y"<<i<<"y"<<j<<" >= 0"<<endl;
                        }
                }
        }

    /*This section prints constraint 7*/

        for (i=1;i<=n;i++)
        {
                f <<"T"<<i<<" - "<<"U"<<i;
                for (j=0;j<=n;j++)
                {
                        if (j!=i)
                        {
                                f <<" + ";
                                f <<"y"<<i<<"y"<<j;
                        }
                }
                a=B-d[i];
                f <<" = "<<a<<endl;
        }

    /*Binary variables are printed*/

        f <<"Binaries"<<endl;
        for (i=0;i<=n;i++)
        {
                for (j=0;j<=n;j++)
                {
                        if (j!=i)
                        {
                                f <<"x"<<i<<"x"<<j<<endl;
                        }
                }
```

```
            }
            f <<"End";
            f.close();
      }
}

/*Integer random number generator*/

      int rnd(int range)
      {
            int y;
            y=rand()%range;
            return(y);
      }

/*Randon number seed generator*/

      void seedrnd(void)
      {
            srand(seed);
      }
```

APPENDIX B – NETWORK TARDINESS PROBLEM BUILDING C CODE

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include <time.h>

#define RANGE1 10
#define RANGE2 10
#define seed 10


/*******************************************************************

THIS PROGRAM OUTPUTS MATHEMATICAL NETWORK PROGRAMMING FORMULATIONS FOR
TARDINESS SCHEDULING PROBLEMS WITH SEQUENCE DEPENDENT SETUPS, BASED ON
AN INPUT NUMBER OF JOBS, MACHINES, DUE DATE RANGE, AND NUMBER OF
PROBLEMS TO BE CREATED.

*******************************************************************

Variables:
n - Number of jobs (input)
np - Number of problems to be output (input)
i,j,k - Loop indexes
x - Counter
s - Setup times
p - Processing times
d - Due dates/times
B - A sufficiently large number
mc - Maximum changeover time
m - Number of machines
a - Temporary value for B - d (large number minus due date)
ddr - Maximum value of due date range
RANGE1 - Maximum of the range of random processing times
RANGE2 - Maximum of the range of random changeover times
*/

int rnd(int range);
void seedrnd(void);

void main()

{
        int n,np,i,j,k,x,s[20][20],p[20],d[20],B,mc,m,a,ddr;

        seedrnd();

/*Input the number of jobs and problems*/

        cout <<"Enter the number of jobs:";
        cin >>n;

        cout <<"Enter the number of machines:";
```

```
        cin >>m;

        cout <<"Enter the due date range:";
        cin >>ddr;

        cout <<"Enter the number of problems:";
        cin >>np;

        for (k=1;k<=np;k++)

/*This section creates an output file name based on the number of jobs
and an index number*/

        {
                char name[16]="a:tarm";
                char buffer1[5];
                char buffer2[5];
                char buffer3[5];
                char ext[5]=".lp";

                itoa (n,buffer1,10);
                strcat(name,buffer1);
                itoa (m,buffer3,10);
                strcat(name,buffer3);
                itoa (k,buffer2,10);
                strcat(name,buffer2);
                strcat(name,ext);

                fstream f;
                f.open(name,ios::out);

/*Generates processing times, setup times, due dates, and B*/

                B=0;
                for (i=1;i<=n;i++)
                {
                        p[i]=rnd(RANGE1)+1;
                        B=B+p[i];
                        d[i]=rnd(ddr)+1;
                        mc=0;
                        for (j=0;j<=n;j++)
                        {
                                if (i!=j)
                                {
                                        s[j][i]=rnd(RANGE2)+1;
                                        if (s[j][i]>mc)
                                        {
                                                mc=s[j][i];
                                        }
                                }
                        }
                        B=B+mc;
                }
                B=200;

/*Prints constants
```

```
        f <<"processing times: ";
        for (i=1;i<=n;i++)
        {
                f <<p[i]<<"   ";
        }
        f <<endl;
        f <<"due dates: ";
        for (i=1;i<=n;i++)
        {
                f <<d[i]<<"   ";
        }
        f <<endl;
        f <<"setup times:"<<endl;
        for (i=0;i<=n;i++)
        {
                for (j=1;j<=n;j++)
                {
                        if (i!=j)
                        {
                                f <<s[i][j]<<"   ";
                        }
                        else
                        {
                                f <<"      ";
                        }
                }
                f <<endl;
        }

/*This section prints the objective function*/

        f <<"Minimize"<<endl;
        f <<"T1";
        for (i=2;i<=n;i++)
        {
                f <<" + T"<<i;
        }
        f <<endl;

/*This section prints constraint 1*/

        f <<"Subject To"<<endl;
        for (i=1;i<=n;i++)
        {
                x=0;
                for (j=0;j<=n;j++)
                {
                        if (j!=i)
                        {
                                if (x>0)
                                {
                                        f <<" + ";
                                }
                                f <<"x"<<i<<"x"<<j;
                                x=x++;
                        }
                }
```

```
        f <<" = 1"<<endl;
        }


/*This section prints constraint 2*/

        for (j=1;j<=n;j++)
        {
                x=0;
                for (i=0;i<=n;i++)
                {
                        if (i!=j)
                        {
                                if (x>0)
                                {
                                        f <<" + ";
                                }
                                f <<"x"<<i<<"x"<<j;
                                x=x++;
                        }
                }
        f <<" = 1"<<endl;
        }


/*This section prints constraint 3*/

        f <<"x0x1";
        for (j=2;j<=n;j++)
        {
                f <<" + x0x"<<j;
        }
        f <<" <= "<<m<<endl;


/*This section prints constraint 4*/

        f <<"x1x0";
        for (j=2;j<=n;j++)
        {
                f <<" + x"<<j<<"x0";
        }
        f <<" <= "<<m<<endl;


/*This section prints constraint 5*/

        for (j=1;j<=n;j++)
        {
                x=0;
                for (i=0;i<=n;i++)
                {
                        if (i!=j)
                        {
                                if (x>0)
                                {
                                        f <<" + ";
                                }
                                f <<"y"<<i<<"y"<<j<<" - y"<<j<<"y"<<i;
                                x=x++;
                        }
                }
```

```
                }
                x=0;
                for (i=0;i<=n;i++)
                {
                        if (i!=j)
                        {
                                f <<" - ";
                                f <<s[i][j]<<" x"<<i<<"x"<<j;
                        }
                }
        f <<" = "<<p[j]<<endl;
        }


    /*This section prints constraint 6*/

        for (i=0;i<=n;i++)
        {
                for (j=0;j<=n;j++)
                {
                        if (j!=i)
                        {
                                f <<B<<" x"<<i<<"x"<<j<<" -
y"<<i<<"y"<<j<<" >= 0"<<endl;
                        }
                }
        }


    /*This section prints constraint 7*/

        for (i=1;i<=n;i++)
        {
                f <<"T"<<i;
                for (j=0;j<=n;j++)
                {
                        if (j!=i)
                        {
                                f <<" + ";
                                f <<"y"<<i<<"y"<<j;
                        }
                }
                a=B-d[i];
                f <<" >= "<<a<<endl;
        }


    /*Binary variables are printed*/

        f <<"Binaries"<<endl;
        for (i=0;i<=n;i++)
        {
                for (j=0;j<=n;j++)
                {
                        if (j!=i)
                        {
                                f <<"x"<<i<<"x"<<j<<endl;
                        }
                }
        }
```

```
            f <<"End";
            f.close();
        }
}

/*Integer random number generator*/

        int rnd(int range)
        {
            int y;
            y=rand()%range;
            return(y);
        }

/*Randon number seed generator*/

        void seedrnd(void)
        {
            srand(seed);
        }
```

APPENDIX C – ZHU AND HEADY ET PROBLEM BUILDING C CODE

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include <time.h>

#define RANGE1 10
#define RANGE2 10
#define seed 10

/*******************************************************************

THIS PROGRAM OUTPUTS MATHEMATICAL PROGRAMMING FORMULATIONS FOR ZHU AND
HEADY EARLINESS/TARDINESS SCHEDULING PROBLEMS WITH SEQUENCE DEPENDENT
SETUPS, BASED ON AN INPUT NUMBER OF JOBS, MACHINES, DUE DATE RANGE, AND
NUMBER OF PROBLEMS TO BE CREATED.

*******************************************************************

Variables:
n - Number of jobs (input)
np - Number of problems to be output (input)
i,j,k,l - Loop indexes
x - Counter
s - Setup times
p - Processing times
d - Due dates/times
B - A sufficiently large number
mc - Maximum changeover time
m - Number of machines
w - Temporary value for p + s - B (processing time plus setup time
minus a large number)
ddr - Maximum value of due date range
RANGE1 - Maximum of the range of random processing times
RANGE2 - Maximum of the range of random changeover times
*/

int rnd(int range);
void seedrnd(void);

void main()

{
        int n,np,i,j,k,l,x,s[25][25],p[25],d[25],B,mc,w,m,ddr;

        seedrnd();

/*Input the number of jobs and problems*/

        cout <<"Enter the number of jobs:";
        cin >>n;
```

```
        cout <<"Enter the number of machines:";
        cin >>m;

        cout <<"Enter the due date range:";
        cin >>ddr;

        cout <<"Enter the number of problems:";
        cin >>np;


        for (l=1;l<=np;l++)

/*This section creates an output file name based on the number of jobs
and an index number*/

        {
                char name[16]="a:zhum";
                char buffer1[5];
                char buffer2[5];
                char buffer3[5];
                char ext[5]=".lp";

                itoa (n,buffer1,10);
                strcat(name,buffer1);
                itoa (m,buffer3,10);
                strcat(name,buffer3);
                itoa (l,buffer2,10);
                strcat(name,buffer2);
                strcat(name,ext);

                fstream f;
                f.open(name,ios::out);

/*Generates processing times, setup times, due dates, and B*/

                B=0;
                for (i=1;i<=n;i++)
                {
                        p[i]=rnd(RANGE1)+1;
                        B=B+p[i];
                        d[i]=rnd(ddr)+1;
                        mc=0;
                        for (j=0;j<=n;j++)
                        {
                                if (i!=j)
                                {
                                        s[j][i]=rnd(RANGE2)+1;
                                        if (s[j][i]>mc)
                                        {
                                                mc=s[j][i];
                                        }
                                }
                        }
                        B=B+mc;
                }
                B=300;
```

```
/*Prints constants*

        f <<"processing times: ";
        for (i=1;i<=n;i++)
        {
                f <<p[i]<<"   ";
        }
        f <<endl;
        f <<"due dates: ";
        for (i=1;i<=n;i++)
        {
                f <<d[i]<<"   ";
        }
        f <<endl;
        f <<"setup times:"<<endl;
        for (i=0;i<=n;i++)
        {       for (j=1;j<=n;j++)
                {       if (i!=j)
                        {
                                f <<s[i][j]<<"   ";
                        }
                        else
                        {
                                f <<"      ";
                        }
                }
                f <<endl;
        }

/*This section prints the objective function*/

        f <<"Minimize"<<endl;
        f <<"T1 + U1";
        for (i=2;i<=n;i++)
        {
                f <<" + T"<<i<<" + U"<<i;
        }
        f <<endl;

/*This section prints constraint 1*/

        f <<"Subject To"<<endl;
        for (i=0;i<=n;i++)
        {
                for(k=1;k<=m;k++)
                {
                        f <<"z"<<i<<"z"<<k;
                        for (j=1;j<=n;j++)
                        {
                                if (j!=i)
                                {
                                        f <<" - x"<<i<<"x"<<j<<"x"<<k;
                                }
                        }
                        f <<" >= 0"<<endl;
                }
        }
```

```
/*This section prints constraint 2*/

    for (j=1;j<=n;j++)
    {
        for(k=1;k<=m;k++)
        {
            f <<"z"<<j<<"z"<<k;
            for (i=0;i<=n;i++)
            {
                if (j!=i)
                {
                    f <<" - x"<<i<<"x"<<j<<"x"<<k;
                }
            }
            f <<" = 0"<<endl;
        }
    }


/*This section prints constraint 3*/


    for (i=1;i<=n;i++)
    {
        x=0;
        for(k=1;k<=m;k++)
        {
            if (x>0)
            {
                f <<" + ";
            }
            f <<"z"<<i<<"z"<<k;
            x=x++;
        }
        f <<" = 1"<<endl;
    }

/*This section prints constraint 4*/

    for (j=1;j<=n;j++)
    {
        x=0;
        for (i=0;i<=n;i++)
        {
            if (i!=j)
            {
                for (k=1;k<=m;k++)
                {
                    w = p[j] + s[i][j] - B;
                    f <<"y"<<j<<" - y"<<i<<" - "<<B<<"
x"<<i<<"x"<<j<<"x"<<k<<" >= "<<w<<endl;
                }
            }
        }
    }
```

```
/*This section prints constraint 5*/

        for (i=1;i<=n;i++)
        {
                f <<"y"<<i<<" - T"<<i<<" + U"<<i<<" = "<<d[i]<<endl;
        }

/*Binary variables are printed*/

        f <<"Binaries"<<endl;
        for (i=0;i<=n;i++)
        {
                for (j=1;j<=n;j++)
                {
                        for (k=1;k<=m;k++)
                        {
                                if (j!=i)
                                {
                                        f <<"x"<<i<<"x"<<j<<"x"<<k<<endl;
                                }
                        }
                }
        }
        for (j=0;j<=n;j++)
        {
                for (k=1;k<=m;k++)
                {
                        if (j!=i)
                        {
                                f <<"z"<<j<<"z"<<k<<endl;
                        }
                }
        }
        f <<"End";
        f.close();
    }
}

/*Integer random number generator*/

        int rnd(int range)
        {
                int y;
                y=rand()%range;
                return(y);
        }

/*Randon number seed generator*/

        void seedrnd(void)
        {
                srand(seed);
        }
```

APPENDIX D – ZHU AND HEADY TARDINESS PROBLEM BUILDING

C CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include <time.h>

#define RANGE1 10
#define RANGE2 10
#define seed 10


/*********************************************************************

THIS PROGRAM OUTPUTS MATHEMATICAL PROGRAMMING FORMULATIONS FOR ZHU AND
HEADY TARDINESS SCHEDULING PROBLEMS WITH SEQUENCE DEPENDENT SETUPS,
BASED ON AN INPUT NUMBER OF JOBS, MACHINES, DUE DATE RANGE, AND NUMBER
OF PROBLEMS TO BE CREATED.

*********************************************************************

Variables:
n - Number of jobs (input)
np - Number of problems to be output (input)
i,j,k,l - Loop indexes
x - Counter
s - Setup times
p - Processing times
d - Due dates/times
B - A sufficiently large number
mc - Maximum changeover time
m - Number of machines
w - Temporary value for p + s - B (processing time plus setup time
minus a large number)
ddr - Maximum value of due date range
RANGE1 - Maximum of the range of random processing times
RANGE2 - Maximum of the range of random changeover times
*/

int rnd(int range);
void seedrnd(void);

void main()

{
        int n,np,i,j,k,l,x,s[20][20],p[20],d[20],B,mc,w,m,ddr;

        seedrnd();

/*Input the number of jobs and problems*/
```

```
cout <<"Enter the number of jobs:";
cin >>n;

cout <<"Enter the number of machines:";
cin >>m;

cout <<"Enter the due date range:";
cin >>ddr;

cout <<"Enter the number of problems:";
cin >>np;


for (l=1;l<=np;l++)
```

/*This section creates an output file name based on the number of jobs and an index number*/

```
{
        char name[16]="a:zhut";
        char buffer1[5];
        char buffer2[5];
        char buffer3[5];
        char ext[5]=".lp";

        itoa (n,buffer1,10);
        strcat(name,buffer1);
        itoa (m,buffer3,10);
        strcat(name,buffer3);
        itoa (l,buffer2,10);
        strcat(name,buffer2);
        strcat(name,ext);

        fstream f;
        f.open(name,ios::out);

/*Generates processing times, setup times, due dates, and B*/

        B=0;
        for (i=1;i<=n;i++)
        {
                p[i]=rnd(RANGE1)+1;
                B=B+p[i];
                d[i]=rnd(ddr)+1;
                mc=0;
                for (j=0;j<=n;j++)
                {
                        if (i!=j)
                        {
                                s[j][i]=rnd(RANGE2)+1;
                                if (s[j][i]>mc)
                                {
                                        mc=s[j][i];
                                }
                        }
                }
                B=B+mc;
```

```
        }
        B=200;

/*Prints constants

        f <<"processing times: ";
        for (i=1;i<=n;i++)
        {
                f <<p[i]<<"   ";
        }
        f <<endl;
        f <<"due dates: ";
        for (i=1;i<=n;i++)
        {
                f <<d[i]<<"   ";
        }
        f <<endl;
        f <<"setup times:"<<endl;
        for (i=0;i<=n;i++)
        {
                for (j=1;j<=n;j++)
                {
                        if (i!=j)
                        {
                                f <<s[i][j]<<"   ";
                        }
                        else
                        {
                                f <<"      ";
                        }
                }
                f <<endl;
        }

/*This section prints the objective function*/

        f <<"Minimize"<<endl;
        f <<"T1";
        for (i=2;i<=n;i++)
        {
                f <<" + T"<<i;
        }
        f <<endl;

/*This section prints constraint 1*/

        f <<"Subject To"<<endl;

        for (i=0;i<=n;i++)
        {
                for(k=1;k<=m;k++)
                {
                        f <<"z"<<i<<"z"<<k;
                        for (j=1;j<=n;j++)
                        {
                                if (j!=i)
                                {
```

```
                        f <<" - x"<<i<<"x"<<j<<"x"<<k;
                    }
                }
                f <<" >= 0"<<endl;
            }
        }

    /*This section prints constraint 2*/

        for (j=1;j<=n;j++)
        {
            for(k=1;k<=m;k++)
            {
                f <<"z"<<j<<"z"<<k;
                for (i=0;i<=n;i++)
                {
                    if (j!=i)
                    {
                        f <<" - x"<<i<<"x"<<j<<"x"<<k;
                    }
                }
                f <<" = 0"<<endl;
            }
        }

    /*This section prints constraint 3*/

        for (i=1;i<=n;i++)
        {
            x=0;
            for(k=1;k<=m;k++)
            {
                if (x>0)
                {
                    f <<" + ";
                }
                f <<"z"<<i<<"z"<<k;
                x=x++;
            }
            f <<" = 1"<<endl;
        }

    /*This section prints constraint 4*/

        for (j=1;j<=n;j++)
        {
            x=0;
            for (i=0;i<=n;i++)
            {
                if (i!=j)
                {
                    for (k=1;k<=m;k++)
                    {
                        w = p[j] + s[i][j] - B;
                        f <<"y"<<j<<" - y"<<i<<" - "<<B<<"
x"<<i<<"x"<<j<<"x"<<k<<" >= "<<w<<endl;
                    }
                }
```

```
                    }
                }
            }

        /*This section prints constraint 5*/

            for (i=1;i<=n;i++)
            {
                f <<"y"<<i<<" - T"<<i<<" <= "<<d[i]<<endl;
            }

        /*Binary variables are printed*/

            f <<"Binaries"<<endl;
            for (i=0;i<=n;i++)
            {
                for (j=1;j<=n;j++)
                {
                    for (k=1;k<=m;k++)
                    {
                        if (j!=i)
                        {
                            f <<"x"<<i<<"x"<<j<<"x"<<k<<endl;
                        }
                    }
                }
            }
            for (j=0;j<=n;j++)
            {
                for (k=1;k<=m;k++)
                {
                    if (j!=i)
                    {
                        f <<"z"<<j<<"z"<<k<<endl;
                    }
                }
            }
            f <<"End";
            f.close();
    }
}

/*Integer random number generator*/

    int rnd(int range)
    {
        int y;
        y=rand()%range;
        return(y);
    }

/*Randon number seed generator*/

    void seedrnd(void)
    {
        srand(seed);
    }
```

APPENDIX E – EARLINESS/TARDINESS HEURISTIC C CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include <time.h>

#define RANGE1 10
#define RANGE2 10
#define seed 10


/*******************************************************************

THIS PROGRAM IS A SCHEDULING HEURISTIC FOR THE EARLINESS/TARDINESS
PROBLEM WITH SEQUENCE DEPENDENT SETUPS.  IT FIRST SORTS AND PLACES THE
JOBS BY EARLIEST DUE DATE.  EACH JOB IS THEN PLACED ON THE MACHINE THAT
YIELDS THE LOWEST VALUE OF SETUP TIME PLUS TARDINESS.  IF ANY
SUBSEQUENT JOBS HAVE A DUE DATE WITHIN D + P + S OF THE PREVIOUS JOB,
REORDERING MAY BE DONE.  ALL PERMUTATIONS OF THE JOBS WITH CLOSE DUE
DATES (AS JUST DEFINED) ARE CHECKED FOR MINUMUM TOTAL SETUP
TIMES/MAKESPAN AND PLACED IN THAT ORDER.  THE SETUP TIME TOTAL INCLUDES
THE NEXT JOB FOLLOWING THE SUB-SEQUENCE BEING CHECKED FOR REORDERING.
ONCE THE SEQUENCE IS ESTABLISHED, THE OPTIMAL INSERTION OF IDLE TIME IS
ACCOMPLISHED USING THE GAREY ET AL METHOD.


*******************************************************************

Variables:
n - number of jobs (input)
np - number of problems to be output (input)
nm - number of machines (input)
ddr - due date range (input)
i,j,k - loop indexes
x - counter
s - setup times
p - processing times
d - due dates/times
c - completion time of job
mc - completion time of machine
t - tardiness of job
sm - selected machine
sv - selected value (s+T)
tv - temp value (s+T)
tt - temp tardiness
pj - previous job on same machine
pc - position counter/number of jobs on machine
pos - temp. position of job on machine
jobmp - job at a specific machine and position
jp - job position (temp transformer)
lj - last job on machine
ljob - temp last job on machine
t - tardiness of each job
ctr - counter of previous jobs (beginning)
```

nj - next job to be placed
rng - range of due dates
jir - array of jobs in range
njob - next job in range
bdd - base due date to start range from
pjloc - previous job location
seqet - sub-sequence total ET
et - earliness and tardiness of current job
jobet - job for which et is being calculated
prej - previous job in sub-sequence examined
best - best order of sub-sequence
thisj - temporary job index for current job while determining the
machine's completion time after reordering
lastj - temporary job index for previous job while determining the
machine's completion time after reordering
stp - starting point for permutation check
stpt - moving starting point for each job being checked
spos - starting position
bestet - permutation sequence with best ET
stpos - starting position of job with earliness and another job in
switch range
dev - the earliness or tardiness of each job
temp,temp2,temp3,temp4,temp5,temp6 - temporary exchange values
tp - temp processing time
td - temp due date
currentj - job for which tp is being calculated
lastj - previous job to current on that machine
c - completion time of job
mc - completion time of machine
sm - selected machine
tc - temp completion time
ec - earliest completion
pj - previous job on same machine
lj - last job on machine
nj - next job to be placed
a - target start time for inserting idle time
b - transformed due dates/times for inserting idle time
startt - start time of job
bc - block counter
inc - number of jobs in increase
dec - number of jobs in decrease
first - smallest indexed job of a block
last - largest indexed job of a block
fbj - temp for first job in the block
length - total processing time of job including setup (position
referenced)
mtotal - total ET of current machine
alltotal - total ET of all machines
minpara - minimum parameter for shifting block
para3 - value of third shift stopping rule
para2 - value of second shift stopping rule
tpara2 - temp value for stopping rule 2
ljpb - last job of previous block
shift - 0 means job does not start at or before target start time, 1
means it does
pone - one job previous to last one placed on same machine
RANGE1 - Maximum of the range of random processing times

```
RANGE2 - Maximum of the range of random changeover times
*/

int rnd(int range);
void seedrnd(void);

void main()

{
        time_t start,finish;
        double elapsed_time;
        int
n,np,nm,nj,ddr,g,h,i,j,k,l,s[100][100],p[100],tp[100],d[100],td[100],jo
b[100];
        int
temp,temp2,temp3,temp4,temp5,temp6,sm,pj,lj[10],jp,ljob,ctr,rng,njob;
        int
t[100],tt,tv,sv,ct[100],mc[10],pc[100],pos,et[100],jobmp[10][100],jir[1
0];
        int
seqms,jobet,prej,best[10],thisj,lastj,complete,inrng,order[100];
        int
prevj,pjloc,temp7,j1,j2,j3,bestms,eata,reorder,count,currentj,pone;
        int
        startt[100],bc,inc[100],dec[100],first[100],last[100],fbj,tpara2,
shift[100];
        int
b[100],a[100],c[100],length[100],dev,alltotal,mtotal,minpara,para3,para
2,ljpb;

        seedrnd();

/*Input the number of jobs and problems*/

        cout <<"Enter the number of jobs:";
        cin >>n;

        cout <<"Enter the number of machines:";
        cin >>nm;

        cout <<"Enter the due date range:";
        cin >>ddr;

        cout <<"Enter the number of problems:";
        cin >>np;

        for (k=1;k<=np;k++)

/*This section creates an output file name based on the number of jobs
and an index number*/

        {
                char name[20]="a:MSnEwP";
                char buffer1[5];
                char buffer2[5];
                char buffer3[5];
                char ext[5]=".txt";
```

```
              itoa (n,buffer1,10);
              strcat(name,buffer1);
              itoa (nm,buffer3,10);
              strcat(name,buffer3);
              itoa (k,buffer2,10);
              strcat(name,buffer2);
              strcat(name,ext);

              fstream f;
              f.open(name,ios::out);

      /*Generates processing times, setup times, due dates, and job
indexes*/

              for (i=1;i<=n;i++)
              {
                      job[i]=i;
                      order[i]=i;
                      p[i]=rnd(RANGE1)+1;
                      d[i]=rnd(ddr)+1;
                      for (j=0;j<=n;j++)
                      {
                              if (i!=j)
                              {
                                      s[j][i]=rnd(RANGE2)+1;
                              }
                      }
              }

      /*Prints constants*/

              f <<"pre-data check"<<endl;
              f <<"job index: ";
              for (i=1;i<=n;i++)
              {
                      f <<job[i]<<"  ";
              }
              f <<endl;
              f <<"job order: ";
              for (i=1;i<=n;i++)
              {
                      f <<order[i]<<"  ";
              }
              f <<endl;
              f <<"processing times: ";
              for (i=1;i<=n;i++)
              {
                      f <<p[i]<<"  ";
              }
              f <<endl;
              f <<"due dates: ";
              for (i=1;i<=n;i++)
              {
                      f <<d[i]<<"  ";
              }
              f <<endl;
```

```
f <<"setup times:"<<endl;
for (i=0;i<=n;i++)
{
        for (j=1;j<=n;j++)
        {
                if (i!=j)
                {
                        f <<s[i][j]<<"   ";
                }
                else
                {
                        f <<"       ";
                }
        }
        f <<endl;
}
f <<endl;

time(&start);

/*Small to large sort of due dates*/

for(i=1;i<=n;i++)
{
        td[i]=d[i];
        tp[i]=p[i];
}

for(i=1;i<=n;i++)
{
        for(j=(i+1);j<=n;j++)
        {
                if(td[j]<td[i])
                {
                        temp=order[i];
                        order[i]=order[j];
                        order[j]=temp;
                        temp2=td[j];
                        td[j]=td[i];
                        td[i]=temp2;
                        temp3=tp[j];
                        tp[j]=tp[i];
                        tp[i]=temp3;
                }
                if(td[j]==td[i])
                {
                        if (tp[j]<tp[i])
                        {
                                temp4=order[i];
                                order[i]=order[j];
                                order[j]=temp4;
                                temp5=td[j];
                                td[j]=td[i];
                                td[i]=temp5;
                                temp6=tp[j];
                                tp[j]=tp[i];
                                tp[i]=temp6;
```

```
                                                }
                                        }
                                }
                        }

        /*Order Check*/

                f <<"job order: ";
                for (i=1;i<=n;i++)
                {
                        f <<order[i]<<"   ";
                }
                f <<endl<<endl;

        /*Initialize machine completion times, last job assigned, job to
machine assignments, and machine position counter*/

                for (i=1;i<=nm;i++)
                {
                        mc[i]=0;
                        lj[i]=0;
                        pc[i]=0;
                }
                for (i=1;i<=nm;i++)
                {
                        jobmp[i][0]=0;
                }

        /*Place jobs on machines*/

                f <<"Placement of jobs on machines:"<<endl;
                for (i=1;i<=n;i++)
                {
                        nj=order[i];
                        f <<i<<". Placement "<<nj<<": ";

                /*Determine machine for placement (minimum s+T)*/

                        for (j=1;j<=nm;j++)
                        {
                                pj=lj[j];
                                if (j==1)
                                {
                                        sm=j;
                                        ct[nj]=mc[j]+p[nj]+s[pj][nj];
                                        t[nj]=ct[nj]-d[nj];
                                        et[nj]=t[nj];
                                        if (t[nj]<0)
                                        {       t[nj]=0;}
                                        sv=t[nj]+s[pj][nj];
                                }
                                else
                                {
                                        tt=mc[j]+p[nj]+s[pj][nj]-d[nj];
                                        if (tt<0)
                                        {
                                                tt=0;
```

```
                                 }
                                 tv=tt+s[pj][nj];
                                 if (tv<sv)
                                 {
                                        sm=j;
                                        ct[nj]=mc[j]+p[nj]+s[pj][nj];
                                        t[nj]=ct[nj]-d[nj];
                                        et[nj]=t[nj];
                                        if (t[nj]<0)
                                        {
                                                t[nj]=0;
                                        }
                                        sv=t[nj]+s[pj][nj];
                                 }
                          }
                   }
                   pc[sm]=pc[sm]++;
                   pos=pc[sm];
                   jobmp[sm][pos]=nj;
                   lj[sm]=nj;
                   mc[sm]=mc[sm]+p[nj]+s[pj][nj];
                   f <<"The sequence on machine "<<sm<<" is: ";
                   for (j=1;j<=pos;j++)
                   {
                          f <<jobmp[sm][j]<<" ";
                   }
                   f <<endl;

            /* Check jobs for shifting while placing */

                   if(pos>2)   /*Does another job exist?*/
                   {
                          if(pos>6)    /*If so, are there more than 5 on
this machine?*/
                          {
                                 ctr=pos-5;  /* If so, start 4 jobs before
current job*/
                          }

                          else
                          {
                                 ctr=1;       /*If not, start from the
first job*/
                          }
                          for (j=ctr;j<pos-1;j++)
                          {
                                 ljob=jobmp[sm][j];       /*Identify job in
that position*/

                                 inrng=1;
                                 if(j==1)    /*If so, is it the first job
on the machine*/
                                 {
                                        pjloc=0;
                                        prevj=0;
                                 }
                                 else
                                 {
```

```
                                        pjloc=j-1;
                                        prevj=jobmp[sm][pjloc];
                        }
                        jir[0]=prevj;
                        jir[1]=ljob;
                        rng=d[ljob]+s[prevj][ljob]+p[ljob]; /*If
earliness, set the range of jobs to look at*/
                        pone=jobmp[sm][pos-1];
                        if(d[pone]<=rng)
                        {
                                for(h=j+1;h<=pos;h++)     /*Look at
all remaining jobs in range*/
                                {
                                        njob=jobmp[sm][h];
                                        inrng=inrng++;
                                        jir[inrng]=njob;
                                }
                                inrng=inrng-1;

                        /*determine min makespan order of jobs in
range*/

                                if(inrng>1) /*generate
permutations of jobs first thought last*/
                                {
                                        bestms=10000;
                                        reorder=1;
                                        for(g=1;g<=inrng;g++)
                                        {
                                                temp7=jir[1];
                                                jir[1]=jir[g];
                                                jir[g]=temp7;
                                                if(inrng<3)
                                                {

        for(l=1;l<=inrng;l++)
                                                        {
                                                                f
<<jir[l]<<" ";
                                                        }
                                                        f <<endl;

                                                        /*Calculate min
setups of jobs in sequence*/

                                                        seqms=0;

        for(l=1;l<=inrng+1;l++)
                                                        {

        jobet=jir[l];
                                                                prej=jir[l-
1];

        eata=s[prej][jobet];

        seqms=seqms+eata;
```

```
MS is "<<bestms<<endl;

is "<<seqms<<endl;

        /*Keep best sequence*/

        bestms=seqms;
best sequence is: ";
        for(l=1;l<=inrng;l++)

        best[l]=jir[l];
<<best[l]<<" ";

MS is "<<bestms<<endl;

        for(j1=2;j1<=inrng;j1++)

        temp7=jir[2];

        jir[2]=jir[j1];

        jir[j1]=temp7;

        for(l=1;l<=inrng;l++)

        f <<jir[l]<<" ";
<<endl;

        /*Calculate min setups of jobs in sequence*/

        seqms=0;
        for(l=1;l<=inrng+1;l++)

        jobet=jir[l];
```

```
                f <<"Best
}
f <<"sequence MS
if(seqms<bestms)
{
                f <<"The
                {
                        f
}
f <<"best
            }
}
if(inrng>2)
{
                {
                if(inrng<4)
                {
                    {
                    }
                    f
                    {
```

```
        prej=jir[1-1];

        eata=s[prej][jobet];

        seqms=seqms+eata;

        f <<"Best MS is "<<bestms<<endl;
                                                                    }
                                                                    f
<<"sequence MS is "<<seqms<<endl;

        if(seqms<bestms)   /*Keep best sequence*/
                                                                    {

        bestms=seqms;

        f <<"The best sequence is: ";

        for(l=1;l<=inrng;l++)

        {

        best[l]=jir[l];

        f <<best[l]<<" ";

        }

        f <<"best MS is "<<bestms<<endl;
                                                                    }
                                                                    }
                                                                    if(inrng>3)
                                                                    {

        for(j2=3;j2<=inrng;j2++)
                                                                    {

        temp7=jir[3];

        jir[3]=jir[j2];

        jir[j2]=temp7;

        if(inrng<5)

        {

        for(l=1;l<=inrng;l++)

        {

                f <<jir[l]<<" ";

        }

        f <<endl;
```

```
/*Calculate min setups of jobs in sequence*/

seqms=0;

for(l=1;l<=inrng+1;l++)

{

        jobet=jir[l];

        prej=jir[l-1];

        eata=s[prej][jobet];

        seqms=seqms+eata;

        f <<"Best MS is "<<bestms<<endl;

}

f <<"sequence MS is "<<seqms<<endl;

if(seqms<bestms)  /*Keep best sequence*/

{

        bestms=seqms;

        f <<"The best sequence is: ";

        for(l=1;l<=inrng;l++)

        {

                best[l]=jir[l];

                f <<best[l]<<" ";

        }

        f <<"best MS is "<<bestms<<endl;

}

}

if(inrng>4)

{

for(j3=4;j3<=inrng;j3++)

{
```

```
temp7=jir[4];

jir[4]=jir[j3];

jir[j3]=temp7;

for(l=1;l<=inrng;l++)

{

        f <<jir[l]<<" ";

}

f <<endl;

/*Calculate min setups of jobs in sequence*/

seqms=0;

for(l=1;l<=inrng+1;l++)

{

        jobet=jir[l];

        prej=jir[l-1];

        eata=s[prej][jobet];

        seqms=seqms+eata;

        f <<"Best MS is "<<bestms<<endl;

}

f <<"sequence MS is "<<seqms<<endl;

if(seqms<bestms)   /*Keep best sequence*/

{

        bestms=seqms;

        f <<"The best sequence is: ";

        for(l=1;l<=inrng;l++)

        {

                best[l]=jir[l];

                f <<best[l]<<" ";
```

```
                    }

                    f <<"best MS is "<<bestms<<endl;

               }

          temp7=jir[4];

          jir[4]=jir[j3];

          jir[j3]=temp7;

     }

     }

     temp7=jir[3];

     jir[3]=jir[j2];

     jir[j2]=temp7;
                                                  }
                                             }

     temp7=jir[2];

     jir[2]=jir[j1];

     jir[j1]=temp7;
                                        }
                                   }
                                   temp7=jir[1];
                                   jir[1]=jir[g];
                                   jir[g]=temp7;
                              }
                         }
                         else
                         {
                              reorder=0;
                         }

               /*Put best order into machine sequence*/

                         if(reorder==1)
                         {
                              for(l=1;l<=inrng;l++)
                              {
                                   jobmp[sm][j+l-
1]=best[l];
                              }
                              jp=pc[sm];
                              lj[sm]=jobmp[sm][jp];
                              complete=0;
                              for(l=1;l<=jp;l++)
                              {
                                   lastj=jobmp[sm][l-1];
```

```
                                                    thisj=jobmp[sm][l];

          complete=complete+p[thisj]+s[lastj][thisj];
                                              }
                                              mc[sm]=complete;
                                              f <<"The sequence on machine
"<<sm<<" is: ";

                                              for (l=1;l<=pos;l++)
                                              {
                                                      f <<jobmp[sm][l]<<" ";
                                              }
                                              f <<endl;
                                              j=pos;
                                    }
                              }
                        }
                  }
          }

     /*Assignments*/

          f <<"Job to machine assignments: "<<endl;
          for (i=1;i<=nm;i++)
          {
                  f <<"Machine "<<i<<":";
                  for (j=1;j<=pc[i];j++)
                  {
                          f <<" "<<jobmp[i][j];
                  }
                  f <<endl;
          }

     /*Insert idle time */

          alltotal=0;
          for (i=1;i<=nm;i++)        /*insert idle time for each machine
seperately*/
          {
                  count=pc[i];

          /*Initialize indexes, times, and counters*/

                  for (j=1;j<=count;j++)
                  {
                          currentj=jobmp[i][j];
                          lastj=jobmp[i][j-1];
                          length[j]=p[currentj]+s[lastj][currentj];
                          b[j]=d[currentj];
                          a[j]=b[j]-length[j];
                  }

                  bc=0; /*initialize block counter*/
                  for (j=1;j<=n;j++)        /*initialize inc and dec*/
                  {
                          inc[j]=0;
                          dec[j]=0;
                          shift[j]=0;
```

```
                              }
                              startt[0]=0;
                              length[0]=0;
                              for (j=1;j<=count;j++)
                              {
                                      if(startt[j-1]+length[j-1]<a[j])     /*no
discrepancy*/

                                      {
                                              startt[j]=a[j];
                                              shift[j]=1;
                                              bc=bc++;
                                              inc[bc]=1;
                                              dec[bc]=0;
                                              first[bc]=j;
                                              last[bc]=j;
                                      }
                                      else if(startt[j-1]+length[j-1]==a[j])     /*no
discrepancy*/

                                      {
                                              startt[j]=startt[j-1]+length[j-1];
                                              last[bc]=j;
                                              inc[bc]=inc[bc]++;
                                              shift[j]=1;
                                      }
                                      else  /*positive discrepancy*/
                                      {
                                              startt[j]=startt[j-1]+length[j-1];
                                              if(bc==0)
                                              {
                                                      bc=bc++;
                                                      first[bc]=j;
                                              }
                                              dec[bc]=dec[bc]++;
                                              last[bc]=j;
                                              fbj=first[bc];
                                              if(dec[bc]==inc[bc] && startt[fbj]!=0)
        /*shift block*/
                                              {
                                                      /*Find ammount to shift - find min
of 3 stopping rules*/
                                                      minpara=startt[fbj];     /*first
stopping rule*/
                                                      if(bc>1)
                                                      {
                                                              ljpb=last[bc-1];   /*third
stopping rule*/
                                                              para3=startt[fbj]-
startt[ljpb]-length[ljpb];
                                                              if(para3<minpara)
                                                              {
                                                                      minpara=para3;
                                                              }
                                                      }
                                                      para2=10000;        /*second stopping
rule*/
                                                      for(l=first[bc];l<=last[bc];l++)
                                                      {
```

```
                                if(shift[l]==0)
                                {
                                        tpara2=startt[l]-a[l];
                                        if(tpara2<para2 &&
tpara2>=0)

                                        {
                                                para2=tpara2;
                                        }
                                }
                        }
                        if(para2<minpara)
                        {
                                minpara=para2;
                                dec[bc]=dec[bc]--;
                                inc[bc]=inc[bc]++;
                        }

                        /*Shift block*/
                        for(l=first[bc];l<=last[bc];l++)
                        {
                                startt[l]=startt[l]-minpara;
                                if(startt[l]<=a[l])
                                {
                                        shift[l]=1;
                                }
                        }
                        if(bc>1)
                        {

        if(startt[fbj]==startt[ljpb]+length[ljpb])
                                {
                                        last[bc-1]=last[bc];
                                        inc[bc-1]=inc[bc-
1]+inc[bc];

                                        dec[bc-1]=dec[bc-
1]+dec[bc];

                                        bc=bc-1;
                                }
                        }
                }
        }
        mtotal=0;
        for (j=1;j<=count;j++)
        {
                f <<"The start time of job "<<j<<" is
"<<startt[j]<<endl;
                c[j]=startt[j]+length[j];
                f <<"The completion time of job "<<j<<" is
"<<c[j]<<endl;
                dev=c[j]-b[j];
                if(dev<0)
                {
                        dev=b[j]-c[j];
                        mtotal=mtotal+dev;
                }
                else
```

```
                              {
                                      mtotal=mtotal+dev;
                              }
                              f <<"The total ET of this machine is
"<<mtotal<<endl;
                      }
                  alltotal=alltotal+mtotal;
              }
          f <<"The total ET of all machines is "<<alltotal<<endl;

      /*Calculate and print elapsed time*/

          time(&finish);
          elapsed_time=difftime(finish,start);
          f <<"Time = "<<elapsed_time<<" seconds"<<endl;

          f.close();
      }
}

/*Integer random number generator*/

      int rnd(int range)
      {
          int y;

          y=rand()%range;
          return(y);
      }

/*Randon number seed generator*/

      void seedrnd(void)
      {
          srand(seed);
      }
```

APPENDIX F – EARLIEST DUE DATE ET HEURISTIC C CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include <time.h>

#define RANGE1 10
#define RANGE2 10
#define seed 10


/************************************************************************

THIS PROGRAM IS A SCHEDULING HEURISTIC WHICH SORTS JOBS BY EDD, PLACES
THEM ON MACHINES IN MACHINE ORDER (BY EDD), AND INSERTS OPTIMAL IDLE
TIME FOR THE ET PROBLEM WITH SEQUENCE DEPENDENT SETUP TIMES

************************************************************************


Variables:
n - number of jobs (input)
np - number of problems to be output (input)
nm - number of machines (input)
ddr - due date range (input)
i,j,k,l - loop indexes
s - setup times
p - processing times
job - job index numbers
order - EDD order of jobs
count - temp value for machine position counter
nj - temp index number of job based on EDD order
pos - temp value for machine position counter
pc - position counter for each machine
jobmp - the job placed on a specific machine and position
dev - the earliness or tardiness of each job
temp,temp2,temp3,temp4,temp5,temp6 - temporary exchange values
tp - temp processing time
td - temp due date
currentj - job for which tp is being calculated
lastj - previous job to current on that machine
d - due times/dates
c - completion time of job
mc - completion time of machine
sm - selected machine
lj - last job on machine
a - target start time to insert idle time
b - due date/time after transformed to insert idle time
startt - start time of job
bc - block counter
inc - number of jobs in increase
dec - number of jobs in decrease
first - smallest indexed job of a block
last - largest indexed job of a block
```

```
fbj - temp for first job in the block
length - total processing time of job including setup (position
referenced)
mtotal - total ET of current machine
alltotal - total ET of all machines
minpara - minimum parameter for shifting block
para3 - value of third shift stopping rule
para2 - value of second shift stopping rule
tpara2 - temp value for stopping rule 2
ljpb - last job of previous block
shift - 0 means job does not start at or before target start time, 1
means it does
RANGE1 - Maximum of the range of random processing times
RANGE2 - Maximum of the range of random changeover times
*/


int rnd(int range);
void seedrnd(void);


void main()


{
        time_t start,finish;
        double elapsed_time;
        int
n,np,nm,nj,ddr,i,j,k,l,s[100][100],p[100],tp[100],d[100],td[100],job[10
0];
        int
temp,temp2,temp3,temp4,temp5,temp6,sm,lj[10],count,currentj,lastj;
        int pc[10],mc[10],pos,jobmp[10][100],order[100];
        int
        startt[100],bc,inc[100],dec[100],first[100],last[100],fbj,tpara2,
shift[100];
        int
b[100],a[100],c[100],length[100],dev,alltotal,mtotal,minpara,para3,para
2,ljpb;

        seedrnd();

/*Input the number of jobs and problems*/

        cout <<"Enter the number of jobs:";
        cin >>n;

        cout <<"Enter the number of machines:";
        cin >>nm;

        cout <<"Enter the due date range:";
        cin >>ddr;

        cout <<"Enter the number of problems:";
        cin >>np;

        for (k=1;k<=np;k++)

/*This section creates an output file name based on the number of jobs
and an index number*/
```

```
        {
                char name[20]="a:eddb";
                char buffer1[5];
                char buffer2[5];
                char buffer3[5];
                char ext[5]=".txt";

                itoa (n,buffer1,10);
                strcat(name,buffer1);
                itoa (nm,buffer3,10);
                strcat(name,buffer3);
                itoa (k,buffer2,10);
                strcat(name,buffer2);
                strcat(name,ext);

                fstream f;
                f.open(name,ios::out);

        /*Generates processing times, setup times, due dates, and job
indexes*/

                for (i=1;i<=n;i++)
                {
                        job[i]=i;
                        order[i]=i;
                        p[i]=rnd(RANGE1)+1;
                        d[i]=rnd(ddr)+1;
                        for (j=0;j<=n;j++)
                        {
                                if (i!=j)
                                {
                                        s[j][i]=rnd(RANGE2)+1;
                                }
                        }
                }

        /*Prints constants*/

                f <<"job index: ";
                for (i=1;i<=n;i++)
                {
                        f <<job[i]<<"   ";
                }
                f <<endl;

                f <<"processing times: ";
                for (i=1;i<=n;i++)
                {
                        f <<p[i]<<"   ";
                }
                f <<endl;
                f <<"due dates: ";
                for (i=1;i<=n;i++)
                {
                        f <<d[i]<<"   ";
```

```
        }
        f <<endl;
        f <<"setup times:"<<endl;
        for (i=0;i<=n;i++)
        {
                for (j=1;j<=n;j++)
                {
                        if (i!=j)
                        {
                                f <<s[i][j]<<"   ";
                        }
                        else
                        {
                                f <<"      ";
                        }
                }
                f <<endl;
        }
        f <<endl;

        time(&start);

/*Small to large sort of due dates*/

        for(i=1;i<=n;i++)
        {
                td[i]=d[i];
                tp[i]=p[i];
        }

        for(i=1;i<=n;i++)
        {
                for(j=(i+1);j<=n;j++)
                {
                        if(td[j]<td[i])
                        {
                                temp=order[i];
                                order[i]=order[j];
                                order[j]=temp;
                                temp2=td[j];
                                td[j]=td[i];
                                td[i]=temp2;
                                temp3=tp[j];
                                tp[j]=tp[i];
                                tp[i]=temp3;
                        }
                        if(td[j]==td[i])
                        {
                                if (tp[j]<tp[i])
                                {
                                        temp4=order[i];
                                        order[i]=order[j];
                                        order[j]=temp4;
                                        temp5=td[j];
                                        td[j]=td[i];
                                        td[i]=temp5;
                                        temp6=tp[j];
```

```
                                            tp[j]=tp[i];
                                            tp[i]=temp6;
                                    }
                            }
                    }
            }

    /*Sequence*/

            f <<"job order: ";
            for (i=1;i<=n;i++)
            {
                    f <<order[i]<<"   ";
            }
            f <<endl<<endl;

    /*Initialize machine completion times, last job assigned, and job
to machine assignments*/

            for (i=1;i<=nm;i++)
            {
                    mc[i]=0;
                    lj[i]=0;
                    pc[i]=0;
            }

            for (i=1;i<=nm;i++)
            {
                            jobmp[i][0]=0;
            }

    /*Place jobs on machines*/

            sm=1;
            for (i=1;i<=n;i++)
            {
                    nj=order[i];

            /*Determine machine for placement (minimum s+T)*/

                    if(sm>nm)
                    {
                            sm=1;
                    }
                    pc[sm]=pc[sm]++;
                    pos=pc[sm];
                    jobmp[sm][pos]=nj;
                    sm=sm++;
            }

    /*Assignments*/

            f <<"Job to machine assignments: "<<endl;
            for (i=1;i<=nm;i++)
            {
                    f <<"Machine "<<i<<":";
                    for (j=1;j<=pc[i];j++)
```

```
                    {
                            f <<" "<<jobmp[i][j];
                    }
                    f <<endl;
            }

      /*Insert idle time */

            alltotal=0;
            for (i=1;i<=nm;i++)        /*insert idle time for each machine
seperately*/
            {
                    count=pc[i];

            /*Initialize indexes, times, and counters*/

                    for (j=1;j<=count;j++)
                    {
                            currentj=jobmp[i][j];
                            lastj=jobmp[i][j-1];
                            length[j]=p[currentj]+s[lastj][currentj];
                            b[j]=d[currentj];
                            a[j]=b[j]-length[j];
                    }

                    bc=0; /*initialize block counter*/
                    for (j=1;j<=n;j++)         /*initialize inc and dec*/
                    {
                            inc[j]=0;
                            dec[j]=0;
                            shift[j]=0;
                    }
                    startt[0]=0;
                    length[0]=0;
                    for (j=1;j<=count;j++)
                    {
                            if(startt[j-1]+length[j-1]<a[j])     /*no
discrepancy*/
                            {
                                    startt[j]=a[j];
                                    shift[j]=1;
                                    bc=bc++;
                                    inc[bc]=1;
                                    dec[bc]=0;
                                    first[bc]=j;
                                    last[bc]=j;
                            }
                            else if(startt[j-1]+length[j-1]==a[j])     /*no
discrepancy*/
                            {
                                    startt[j]=startt[j-1]+length[j-1];
                                    last[bc]=j;
                                    inc[bc]=inc[bc]++;
                                    shift[j]=1;
                            }
                            else  /*positive discrepancy*/
                            {
```

```
                startt[j]=startt[j-1]+length[j-1];
                if(bc==0)
                {
                        bc=bc++;
                        first[bc]=j;
                }
                dec[bc]=dec[bc]++;
                last[bc]=j;
                fbj=first[bc];
                if(dec[bc]==inc[bc] && startt[fbj]!=0)
                {
                        /*Find ammount to shift - find min of 3 stopping rules*/
                        minpara=startt[fbj];     /*first stopping rule*/
                        if(bc>1)
                        {
                                ljpb=last[bc-1];   /*third stopping rule*/
                                para3=startt[fbj]-startt[ljpb]-length[ljpb];
                                if(para3<minpara)
                                {
                                        minpara=para3;
                                }
                        }
                        para2=10000;        /*second stopping rule*/
                        for(l=first[bc];l<=last[bc];l++)
                        {
                                if(shift[l]==0)
                                {
                                        tpara2=startt[l]-a[l];
                                        if(tpara2<para2 && tpara2>=0)
                                        {
                                                para2=tpara2;
                                        }
                                }
                        }
                        if(para2<minpara)
                        {
                                minpara=para2;
                                dec[bc]=dec[bc]--;
                                inc[bc]=inc[bc]++;
                        }

                        /*Shift block*/
                        for(l=first[bc];l<=last[bc];l++)
                        {
                                startt[l]=startt[l]-minpara;
                                if(startt[l]<=a[l])
                                {
                                        shift[l]=1;
                                }
                        }
```

```
                                        if(bc>1)
                                        {

        if(startt[fbj]==startt[ljpb]+length[ljpb])
                                        {
                                                last[bc-1]=last[bc];
                                                inc[bc-1]=inc[bc-
1]+inc[bc];

                                                dec[bc-1]=dec[bc-
1]+dec[bc];

                                                bc=bc-1;
                                        }
                                }
                        }
                }
        }
                mtotal=0;
                for (j=1;j<=count;j++)
                {
                        f <<"The start time of job "<<j<<" is
"<<startt[j]<<endl;

                        c[j]=startt[j]+length[j];
                        f <<"The completion time of job "<<j<<" is
"<<c[j]<<endl;

                        dev=c[j]-b[j];
                        if(dev<0)
                        {
                                dev=b[j]-c[j];
                                mtotal=mtotal+dev;
                        }
                        else
                        {
                                mtotal=mtotal+dev;
                        }
                        f <<"The total ET of this machine is
"<<mtotal<<endl;
                }
                alltotal=alltotal+mtotal;
        }
        f <<"The total ET of all machines is "<<alltotal<<endl;

    /*Calculate and print elapsed time*/

        time(&finish);
        elapsed_time=difftime(finish,start);
        f <<"Time = "<<elapsed_time<<" seconds"<<endl;

        f.close();
    }
}

/*Integer random number generator*/

    int rnd(int range)
    {
        int y;
        y=rand()%range;
```

```
            return(y);
        }

/*Randon number seed generator*/

        void seedrnd(void)
        {
                srand(seed);
        }
```

REFERENCES

J.H. Ahn and J. H. Hyun, Single facility multi-class job scheduling. Computers and Operations Research, 17, 265-272, (1990).

E. M. Arkin and R. O. Roundy. Weighted-tardiness scheduling on parallel machines with proportional weights. Operations Research 39, 64-81 (1991).

T. Back, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford University Press, New York, (1996).

K. R. Baker. Introduction to sequencing and scheduling, New York, Wiley (1974).

K. R. Baker, Computational Experience with a Sequencing Algorithm Adapted to the Tardiness Problem, GSBA Paper 163, Graduate School of Business Administration, Duke University, Durham, N.C., (1976).

K. R. Baker and J. B. Martin, An Experimental Comparison of Solution Algorithms for the Single-Machine Tardiness Problem, Naval Res. Logist. Quart. 21, 187-199 (1974).

K. R. Baker and G. D. Scudder. Sequencing with earliness and tardiness penalties: A review. Operations Research 38 (1), 22-36 (1990).

N. Balakrishnan, J. J. Kanet, and S. V. Sridharan. Early/tardy scheduling with sequence dependent setups on uniform parallel machines. Computers & Operations Research 26, 127-141, (1999).

J. W. Barnes and L. K. Vanston, Scheduling jobs with linear delay penalties and sequence dependent setup costs. Opns Res. 29, 146-160 (1981).

M. Bellmore and G. L. Nernhauser. The traveling salesman problem: a survey. Opns Res. 16, 538 (1968).

J.D. Blocher, Single machine changeover scheduling. Ph.D. dissertation, Purdue University, (1992).

J.D. Blocher. and S. Chand. A forward branch-and-search algorithm and forecast horizon results for the changeover scheduling problem. European Journal of Operational Research 91, 456-470 (1996).

J. Bruno and P. Downey, Complexity of task sequencing with deadlines, setup times. and changeover costs, Society for Industrial and Applied Mathematics Journal of Computing, 7. 393-404, (1978).

L. Chambers (ed.), Practical Handbook of Genetic Algorithms: Applications, Volume I, CRC Press, Boca Raton, FL (1995).

L. Chambers (ed.), Practical Handbook of Genetic Algorithms: New Frontiers, Volume II, CRC Press, Boca Raton, FL, (1995).

Z. Chen. Scheduling with batch setup times and earliness-tardiness penalties. European Journal of Operational Research 96, 518-537 (1997).

T. C. E. Cheng and M. C. Gupta. Survey of scheduling research involving due date determination decisions. European Journal of Operational Research 38, 156-166, (1989).

B. J. Coleman. Technical Note: A simple model for optimizing the single machine early/tardy problem with sequence-dependent setups. Production and Operations Management 1 (2), 225-228 (1992).

D.G. Conway and M.A. Venkataramanan, Genetic Search and the Dynamic Facility Layout Problem, Computers & Operations Research 21. 955-960, (1994).

L. Davis (ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, (1991).

D. M. Dilts and K. D. Ramsing. Joint lot sizing and scheduling of multiple items with sequence-dependent setup costs. Decision Sci. 20, 120-133 (1989).

W. C. Driscoll and H. Emmons, Scheduling production on one machine with changeover costs. AIIE Trans. 9, 388-395 (1977).

J. Du and J.Y.-T. Leung, Minimizing total tardiness on one machine is NP-hard. Math. Opns Res. 15,483-495 (1990).

H. Emmons, "One-Machine Sequencing to Minimize Certain Functions of Job Tardiness," Opns. Res. 17, 701-715 (1969).

B.B. Flynn, Repetitive lots: the use of a sequence dependent setup time scheduling procedure in group technology and traditional shops. J. Opns Mgmt 7. 203-216, (1987).

B.R. Fox and M.B. McMahon, Genetic Operators for Sequencing Problems, in Foundations of Genetic Algorithms, G.J.E. Rawlins (ed.), Morgan Kaufmann, San Mateo, CA, 284-300, (1991).

P.M. Franca et al., A memetic algorithm for the total tardiness single machine scheduling problem, European Journal of Operational Research 132(1), 224, (2001).

B. Freisleben and P. Merz, New Genetic Local Search Operators for the Traveling Salesman Problem, in Parallel Problem-Solving from Nature 4, H-M. Voigt, W. Ebeling, I. Rechenberg, and H-P. Schwefel (eds.), Springer-Verlag, Berlin, 890-899, (1996).

P. S. Gabbert, D. E. Brown, C. L. Huntley. & P. Markowicz; and D. E. Sappington. A system for learning routes and schedules with genetic algorithms. In Proc. of the Fourth Int. Conf. on Genetic-Algorithms and Their Applications, pp. 43D-436. San Diego, CA (1991).

M. R. Garey, et al. One-processor scheduling with symmetric earliness and tardiness penalties. Mathematics of Operations Research 13 (2), (1988).

J. B. Ghosh and C. E. Wells. Due-dates and early/tardy scheduling on identical parallel machines. Naval Research Logistics 41 (1), 17-32, (1994).

C.R. Glassey, Minimum change-over scheduling of several products on one machine. Operations Research, 16, 342-352, (1968).

J. Grefenstette, R. Gopal, B. Rosmaita and D. Van Gucht, Genetic algorithms for the traveling salesman problem. Proc. Int. Conf. Genetic Algorithms and their Applications, pp. 160-168 (1985).

J.N.D. Gupta, Single machine scheduling with multiple job classes. European Journal of Operational Research, 8, 42-45, (1988).

J. N. D. Gupta and W. P. Darrow, The two-machine sequence dependent flowshop scheduling problem. Eur. J. Opl Res. 24, 439-446 (1986).

S. K. Gupta, N jobs and M machines job-shop problems with sequence dependent set-up times. Int. J. Prod. Res. 20, 643-656 (1982).

A. Homaifar, S. Guan, and G.E. Liepins, A New Approach on the Traveling Salesman Problem by Genetic Algorithms, in Proceedings of 5th International Conference on Genetic Algorithms, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 460-466, (1993).

C.R. Houck, J.A. Joines, and M.G. Kay, Comparison of Genetic Algorithms, Random Restart and Two-Opt Switching for Solving Large Location-Allocation Problems, Computers & Operations Research 23, 587-596, (1996).

ILOG, CPLEX 7.1 User's Manual, March, 2001.

P. Jog, J.Y. Suh and D. Van Gucht, The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem in, Proceedings of 3rd international Conference on Genetic Algorithms, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 110-115, (1991).

J. J. Kanet. Minimizing the average deviation of job completion times about a common due date. Naval Research Logistics Quarterly 28, 643-651, (1981).

S.C. Kim, and P.M. Bobrowski, Impact of sequence-dependent setup time on job shop scheduling performance. International Journal of Production Research, 32(7), 1503-1520, (1994).

Y.D. Kim et al., Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process, European Journal of Operational Research 91(1), 124-143, (1996).

E.L. Lawler, A 'Pseudopolynomial' Algorithm for Sequencing Jobs to Minimize Total Tardiness. Ann. Discrete Math. 1 331-342, (1997).

E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (Eds). The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization; Wiley, Chichester, (1985).

Y.H. Lee and S. Kim, Neural network applications for scheduling jobs on parallel machines, Computers & Industrial Engineering 24(1-4), 227-230 (1993).

Y.H. Lee and M. Pinedo, Scheduling jobs on parallel machines with sequence-dependent setup times, European Journal of Operational Research, 100, 464-474, (1997).

Y.H. Lee, K. Bhaskran, and M. Pinedo, A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transactions, 29(1), 45-52, (1997).

A. G. Lockett and A. P. Muhlemann, A scheduling problem involving sequence dependent changeover times. Opns Res. 20, 895-902 (1972).

K. Mathias and D. Whitley, Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem, in Parallel Problem-Solving from Nature 2, R. Manner and B. Manderick (eds.), Elsevier Science Publishers, Amsterdam, 219-228, (1992).

C. E. Miller, A. W. Tucker, and R. A. Zemlin, Integer programming formulations and traveling salesman problems, J. Assoc, Comput. Mach. 7 [2:4], 326-329, (1960).

I.M. Oliver, D.J. Smith, and J.R.C. Holland, A Study of Permutation Crossover Operators on the Traveling Salesman Problem in Proceedings of the 2nd International Conference on Genetic Algorithms, J.J. Grefenstette (ed.), Lawrence Erlbaum Associates, Hillsdale, NJ, 224-230, (1987).

P. S. Ow and T. E. Morton. The single machine early/tardy problem. Management Science 35 (2), 177-191 (1989).

C. H. Pan et al., A heuristic approach for single-machine scheduling with due dates and class setups, Computers & Operations Research 28(11), 1111, (2001).

Y. Park et al., Scheduling jobs on parallel machines applying neural network and heuristic rules, Computers & Industrial Engineering, 38 (1), 189, (2000).

S.S. Panwalker, R.A. Dudek, and M. L. Smith, Sequencing research and the industrial Scheduling Problem. In Symposium on the theory of Scheduling and Its Applications (Edited by S. E. Elmaghrabyl, 29-38, (1973).

S. Parthasarathy and C. Rajendran, An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times of jobs, International Journal of Production Economics 49, 255-263. (1997).

J. C. Picard and M. Queyranne, The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. Opns Res. 26, 86-110 (1978).

P.W. Poon and J.N. Carter, Genetic Algorithm Crossover Operators for Ordering Applications, Computers & Operations Research 22, 135-147, (1995).

T. Prabhakar, A production scheduling problem with sequencing considerations. Mgmt Sci. 21, 34-42 (1974).

P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, Hybrid Genetic Algorithms for the Traveling Salesman Problem, in Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, R.F. Albrecht, C.R. Reeves and N.C. Steele (eds.), Springer-Verlag, Vienna, 559-566, (1993).

G. L. Ragatz, Scheduling to minimize tardiness on a single machine with sequence dependent setup times. Working Paper, Department of Management, Michigan State University (1989).

C.R. Reeves, A Genetic Algorithm for Flowshop Sequencing, Computers & Operations Research 22, 5-13, (1995).

P.A. Rubin and G.L.Ragatz, Scheduling in a Sequnce Dependent Setup Environment with Genetic Search, Computers and Operations Research, Vol. 22, No. 1, P. 85-99, (1995).

J.M.J Schutten, S.L. van de Velde, and W.H.M. Zijm, Single-machine scheduling with release dates, due dates and family setup times. Management Science, 42(8), 1165-1174, (1996).

H. Singh and J. B. Foster. Production scheduling with sequence dependent setup costs. IIE Trans. 43-49 (1987).

F. Sivrikaya-Serifoglu and G. Ulusoy. Parallel machine scheduling with earliness and tardiness penalties. Computers & Operations Research 26, 773-787 (1999).

T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, and C. Whitley, A Comparison of Genetic Sequencing Operators, in Proceedings of 4th International Conference on Genetic Algorithms, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, 69-76, (1991).

X. Sun, et al., Single-machine scheduling with sequnce dependent setup to minimize total weighted squared tardiness, IIE Transactions 31, 113-124, (1999).

X. Sun and J.S. Noble, An approach to job shop scheduling with sequence-dependent setups, Journal of Manufacturing Systems 18 (6), 416-430, (1999).

K.C. Tan and R. Narasimham, Minimizing Tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach, Omega (25(6), 619-634, (1997).

K.C. Tan et al., A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times, Omega 28 (3), 313, (2000).

S. Thangiah, R. Vinayagamoorthy, and A.V. Gubbi, Vehicle Routing and Time Deadlines Using Genetic and Local Algorithms, in Proceedings of 5th International Conference on Genetic Algorithms, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 506-513, (1993).

N.L.J. Ulder, E.H.L. Aarts, H.-J. Bandelt, P.J.M. Laarhoven, and E. Pesch, Genetic Local Search Algorithms for the Traveling Salesman Problem, in Parallel Problem-Solving from Nature, H-P. Schwefel and R. Manner (eds.), Springer-Verlag, Berlin, 109-116 (1991).

E. Uskup and S.B Smith, A branch-and-bound algorithm for two-stage production sequencing problems, Operations Research 23(1), 118-136, (1975).

D. Whitley, T. Starkweather and D'Ann Fuquay, Scheduling problems and traveling salesman. In Proc. of the Third Int. Conf. on Genetic Algorithms and Their Applications, pp. 133-140. Arlington, VA (1989).

C. A. Yano and Y. Kim. Algorithms for a class of single-machine weighted tardiness and earliness problems. European Journal of Operational Research 52, 167-178, (1991).

M. Zbigniew, Genetic Algorithms + Data Structures = Evolutionary Programs, 3rd rev. and extended ed., Springer-Verlag, Berlin; New York, (1996).

Z. Zhu and R. B. Heady. Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach. Computers & Industrial Engineering 38, 297-305 (2000).

G. Zweig, An Effective Tour Construction and Improvement Procedure for the Traveling Salesman Problem, Operations Research 43, 1049-1057 (1995).